CSIT REPORT

Release rls1908$_2$

May 20, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Report History

FD.io CSIT-1908.2 Report history and per .[ww] revision changes are listed below.

.[ww] Revision	Changes
.21	Improved headers and legend in comparison tables.
.20	Initial version

FD.io CSIT Reports follow CSIT-[yy][mm].[ww] numbering format, with version denoted by concatenation of two digit year [yy] and two digit month [mm], and maintenance revision identified by two digit calendar week number [ww].

1.2 Report Structure

FD.io CSIT-1908.2 report contains system performance and functional testing data of VPP-19.08.2 release. PDF version of this report[1] is available for download.

CSIT-1908.2 report is structured as follows:

1. INTRODUCTION: General introduction to FD.io CSIT-1908.2.
   - **Introduction**: This section.
   - **Test Scenarios Overview**: A brief overview of test scenarios covered in this report.
   - **Physical Testbeds**: Description of physical testbeds.
   - **Test Methodology**: Performance benchmarking and functional test methodologies.
2. VPP PERFORMANCE: VPP performance tests executed in physical FD.io testbeds.
   - **Overview**: Tested logical topologies, test coverage and naming specifics.
   - **Release Notes**: Changes in CSIT-1908.2, added tests, environment or methodology changes, known issues.
   - **Packet Throughput**: NDR, PDR throughput graphs based on results from repeated same test job executions to verify repeatibility of measurements.
   - **Speedup Multi-Core**: NDR, PDR throughput multi-core speedup graphs based on results from test job executions.
   - **Packet Latency**: Latency graphs based on results from test job executions.
   - **Soak Tests**: Long duration soak tests are executed using PLRsearch algorithm.

[1] https://docs.fd.io/csit/rls1908_2/report/_static/archive/csit_rls1908_2.21.pdf

  - **NFV Service Density**: Network Function Virtualization (NFV) service density tests focus on measuring total per server throughput at varied NFV service "packing" densities with vswitch providing host dataplane.
  - **Comparisons**: Performance comparisons between VPP releases and between different testbed types.
  - **Throughput Trending**: References to continuous VPP performance trending.
  - **Test Environment**: Performance test environment configuration.
  - **Documentation**: Pointers to CSIT source code documentation for VPP performance tests.

3. DPDK PERFORMANCE: DPDK performance tests executed in physical FD.io testbeds.
   - **Overview**: Tested logical topologies, test coverage.
   - **Release Notes**: Changes in CSIT-1908.2, known issues.
   - **Packet Throughput**: NDR, PDR throughput graphs based on results from repeated same test job executions to verify repeatibility of measurements.
   - **Packet Latency**: Latency graphs based on results from test job executions.
   - **Comparisons**: Performance comparisons between DPDK releases and between different testbed types.
   - **Throughput Trending**: References to regular DPDK performance trending.
   - **Test Environment**: Performance test environment configuration.
   - **Documentation**: Pointers to CSIT source code documentation for DPDK performance tests.
4. VPP DEVICE: VPP functional tests executed in physical FD.io testbeds using containers.
   - **Overview**: Tested virtual topologies, test coverage and naming specifics;
   - **Release Notes**: Changes in CSIT-1908.2, added tests, environment or methodology changes, known issues.
   - **Integration Tests**: Functional test environment configuration.
   - **Documentation**: Pointers to CSIT source code documentation for VPP functional tests.
5. DETAILED RESULTS: Detailed result tables auto-generated from CSIT test job executions using RF (Robot Framework) output files as sources.
   - **VPP Performance NDR/PDR**: VPP NDR/PDR throughput and latency.
   - **VPP Performance MRR**: VPP MRR throughput.
   - **DPDK Performance**: DPDK Testpmd and L3fwd NDR/PDR throughput and latency.
6. TEST CONFIGURATION: VPP DUT configuration data based on VPP API Test (VAT) Commands History auto-generated from CSIT test job executions using RF output files as sources.
   - **VPP Performance NDR/PDR**: Configuration data.
   - **VPP Performance MRR**: Configuration data.
7. TEST OPERATIONAL DATA: VPP DUT operational data auto-generated from CSIT test job executions using RFoutput files as sources.
   - **VPP Performance NDR/PDR**: VPP *show run* outputs under test load.
8. CSIT FRAMEWORK DOCUMENTATION: Description of the overall FD.io CSIT framework.
   - **Design**: Framework modular design hierarchy.
   - **Test naming**: Test naming convention.
   - **Presentation and Analytics Layer**: Description of PAL CSIT analytics module.

- **CSIT RF Tags Descriptions**: CSIT RF Tags used for test suite and test case grouping and selection.

1.3 Test Scenarios

FD.io CSIT-1908.2 report includes multiple test scenarios of VPP centric applications, topologies and use cases. In addition it also covers baseline tests of DPDK sample applications. Tests are executed in physical (performance tests) and virtual environments (functional tests).

Brief overview of test scenarios covered in this report:

1. **VPP Performance**: VPP performance tests are executed in physical FD.io testbeds, focusing on VPP network data plane performance in NIC-to-NIC switching topologies. Tested across Intel Xeon Haswell and Skylake servers, ARM, Denverton, range of NICs (10GE, 25GE, 40GE) and multi-thread/multi-core configurations. VPP application runs in bare-metal host user-mode handling NICs. TRex is used as a traffic generator.
2. **VPP Vhostuser Performance with KVM VMs**: VPP VM service switching performance tests using vhostuser virtual interface for interconnecting multiple NF-in-VM instances. VPP vswitch instance runs in bare-metal user-mode handling NICs and connecting over vhost-user interfaces to VM instances each running VPP with virtio virtual interfaces. Similarly to VPP Performance, tests are run across a range of configurations. TRex is used as a traffic generator.
3. **VPP Memif Performance with LXC and Docker Containers**: VPP Container service switching performance tests using memif virtual interface for interconnecting multiple VPP-in-container instances. VPP vswitch instance runs in bare-metal user-mode handling NICs and connecting over memif (Slave side) interfaces to more instances of VPP running in LXC or in Docker Containers, both with memif interfaces (Master side). Similarly to VPP Performance, tests are run across a range of configurations. TRex is used as a traffic generator.
4. **DPDK Performance**: VPP uses DPDK to drive the NICs and physical interfaces. DPDK performance tests are used as a baseline to profile performance of the DPDK sub-system. Two DPDK applications are tested: Testpmd and L3fwd. DPDK tests are executed in the same testing environment as VPP tests. DPDK Testpmd and L3fwd applications run in host user-mode. TRex is used as a traffic generator.
5. **VPP Functional**: VPP functional tests are executed in virtual FD.io testbeds, focusing on VPP packet processing functionality, including both network data plane and in-line control plane. Tests cover vNIC-to-vNIC vNIC-to-nestedVM-to-vNIC forwarding topologies. Scapy is used as a traffic generator.

All CSIT test data included in this report is auto- generated from RF (Robot Framework) `output.xml` files produced by LF (Linux Foundation) FD.io Jenkins jobs executed against VPP-19.08.2 release artifacts. References are provided to the original FD.io Jenkins job results and all archived source files.

FD.io CSIT system is developed using two main coding platforms: RF and Python2.7. CSIT-1908.2 source code for the executed test suites is available in CSIT branch rls1908_2 in the directory `./tests/<name_of_the_test_suite>`. A local copy of CSIT source code can be obtained by cloning CSIT git repository - **`git clone https://gerrit.fd.io/r/csit`**.

1.4 Physical Testbeds

All FD.io (Fast Data Input/Ouput) CSIT (Continuous System Integration and Testing) performance test results included in this report are executed on the physical testbeds hosted by LF FD.io project, unless otherwise noted.

Two physical server topology types are used:

- **2-Node Topology**: Consists of one server acting as a System Under Test (SUT) and one server acting as a Traffic Generator (TG), with both servers connected into a ring topology. Used for executing tests that require frame encapsulations supported by TG.
- **3-Node Topology**: Consists of two servers acting as a Systems Under Test (SUTs) and one server acting as a Traffic Generator (TG), with all servers connected into a ring topology. Used for executing tests that require frame encapsulations not supported by TG e.g. certain overlay tunnel encapsulations and IPsec. Number of native Ethernet, IPv4 and IPv6 encapsulation tests are also executed on these testbeds, for comparison with 2-Node Topology.

Current FD.io production testbeds are built with SUT servers based on the following processor architectures:

- Intel Xeon: Skylake Platinum 8180, Haswell-SP E5-2699v3, Cascade Lake Platinum 8280, Cascade Lake 6252N.
- Intel Atom: Denverton C3858.
- ARM: TaiShan 2280, hip07-d05.

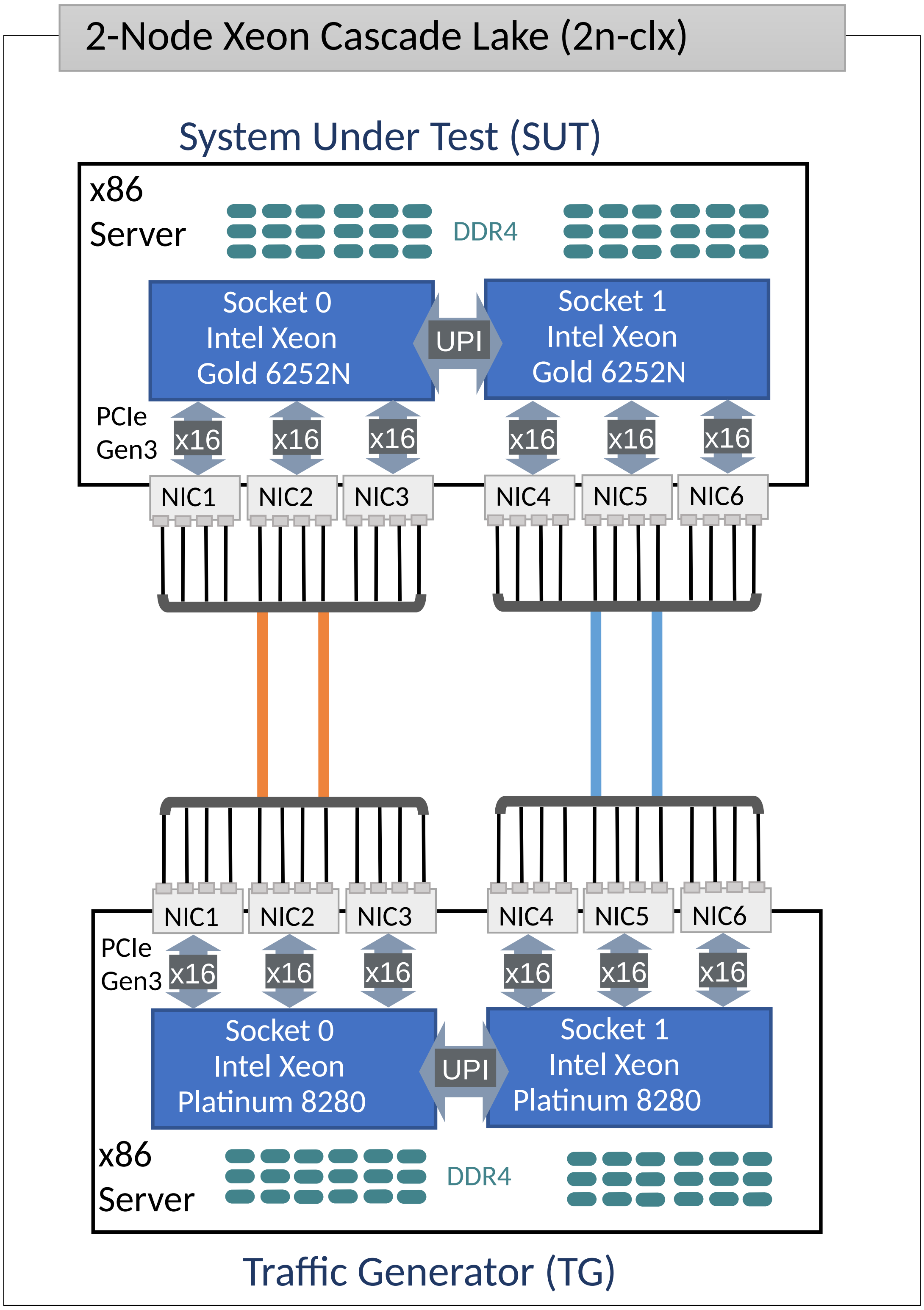
Server SUT performance depends on server and processor type, hence results for testbeds based on different servers must be reported separately, and compared if appropriate.

Complete technical specifications of compute servers used in CSIT physical testbeds are maintained in FD.io CSIT repository: https://git.fd.io/csit/tree/docs/lab/testbed_specifications.md.

Following is the description of existing production testbeds.

1.4.1 2-Node Xeon Cascade Lake (2n-clx)

Three 2n-clx testbeds are in operation in FD.io labs. Each 2n-clx testbed is built with two SuperMicro SYS-7049GP-TRT servers, SUTs are equipped with two Intel Xeon Gold 6252N processors (35.75 MB Cache, 2.30 GHz, 24 cores). TGs are equiped with Intel Xeon Cascade Lake Platinum 8280 processors (38.5 MB Cache, 2.70 GHz, 28 cores). 2n-clx physical topology is shown below.


2-Node Xeon Cascade Lake (2n-clx)
System Under Test (SUT)
x86 Server
DDR4
Socket 0 Intel Xeon Gold 6252N
UPI
Socket 1 Intel Xeon Gold 6252N
PCIe Gen3
x16
NIC1
NIC2
NIC3
NIC4
NIC5
NIC6
Socket 0 Intel Xeon Platinum 8280
Socket 1 Intel Xeon Platinum 8280
Traffic Generator (TG)

SUT servers are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: cx556a-edat ConnectX5 2p100GE Mellanox. (Only testbed t27, t28)
4. NIC-4: empty, future expansion.
5. NIC-5: empty, future expansion.

6. NIC-6: empty, future expansion.

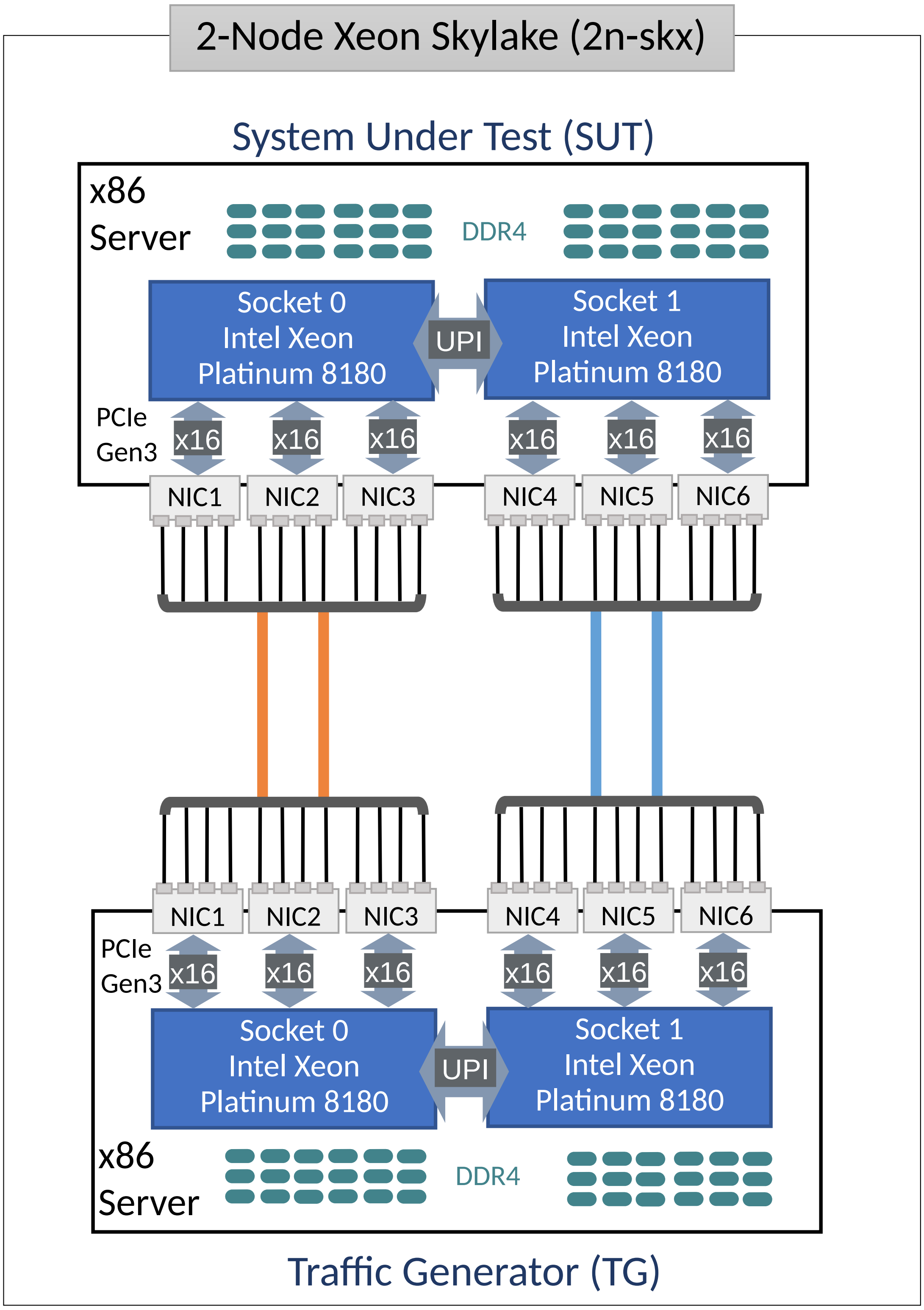
TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: cx556a-edat ConnectX5 2p100GE Mellanox. (Only testbed t27, t28)
4. NIC-4: empty, future expansion.
5. NIC-5: empty, future expansion.
6. NIC-6: x710-DA4 4p10GE Intel. (For self-tests.)

All Intel Xeon Cascade Lake servers run with Intel Hyper-Threading enabled, doubling the number of logical cores exposed to Linux.

1.4.2 2-Node Xeon Skylake (2n-skx)

Four 2n-skx testbeds are in operation in FD.io labs. Each 2n-skx testbed is built with two SuperMicro SYS-7049GP-TRT servers, each in turn equipped with two Intel Xeon Skylake Platinum 8180 processors (38.5 MB Cache, 2.50 GHz, 28 cores). 2n-skx physical topology is shown below.


2-Node Xeon Skylake (2n-skx)
System Under Test (SUT)
x86 Server
DDR4
Socket 0 Intel Xeon Platinum 8180
UPI
Socket 1 Intel Xeon Platinum 8180
PCIe Gen3
x16
NIC1
NIC2
NIC3
NIC4
NIC5
NIC6
PCIe Gen3
Socket 0 Intel Xeon Platinum 8180
UPI
Socket 1 Intel Xeon Platinum 8180
x86 Server
DDR4
Traffic Generator (TG)

SUT servers are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: cx556a-edat ConnectX5 2p100GE Mellanox. (Not used yet.)
4. NIC-4: empty, future expansion.
5. NIC-5: empty, future expansion.

6. NIC-6: empty, future expansion.

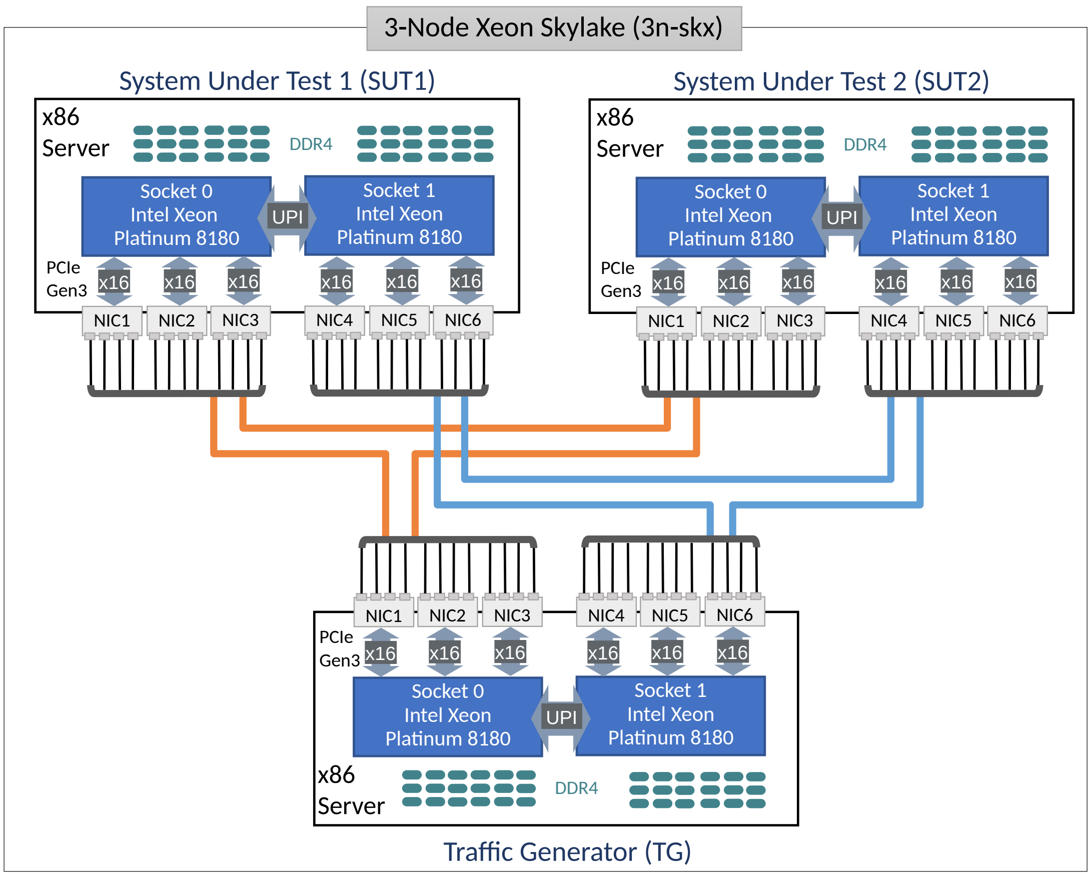
TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: cx556a-edat ConnectX5 2p100GE Mellanox. (Not used yet.)
4. NIC-4: empty, future expansion.
5. NIC-5: empty, future expansion.
6. NIC-6: x710-DA4 4p10GE Intel. (For self-tests.)

All Intel Xeon Skylake servers run with Intel Hyper-Threading enabled, doubling the number of logical cores exposed to Linux, with 56 logical cores and 28 physical cores per processor socket.

1.4.3 3-Node Xeon Skylake (3n-skx)

Two 3n-skx testbeds are in operation in FD.io labs. Each 3n-skx testbed is built with three SuperMicro SYS-7049GP-TRT servers, each in turn equipped with two Intel Xeon Skylake Platinum 8180 processors (38.5 MB Cache, 2.50 GHz, 28 cores). 3n-skx physical topology is shown below.


3-Node Xeon Skylake (3n-skx)
System Under Test 1 (SUT1)
System Under Test 2 (SUT2)
x86 Server
DDR4
Socket 0 Intel Xeon Platinum 8180
UPI
Socket 1 Intel Xeon Platinum 8180
PCIe Gen3
x16
NIC1
NIC2
NIC3
NIC4
NIC5
NIC6
Traffic Generator (TG)

SUT1 and SUT2 servers are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: empty, future expansion.
4. NIC-4: empty, future expansion.

5. NIC-5: empty, future expansion.
6. NIC-6: empty, future expansion.

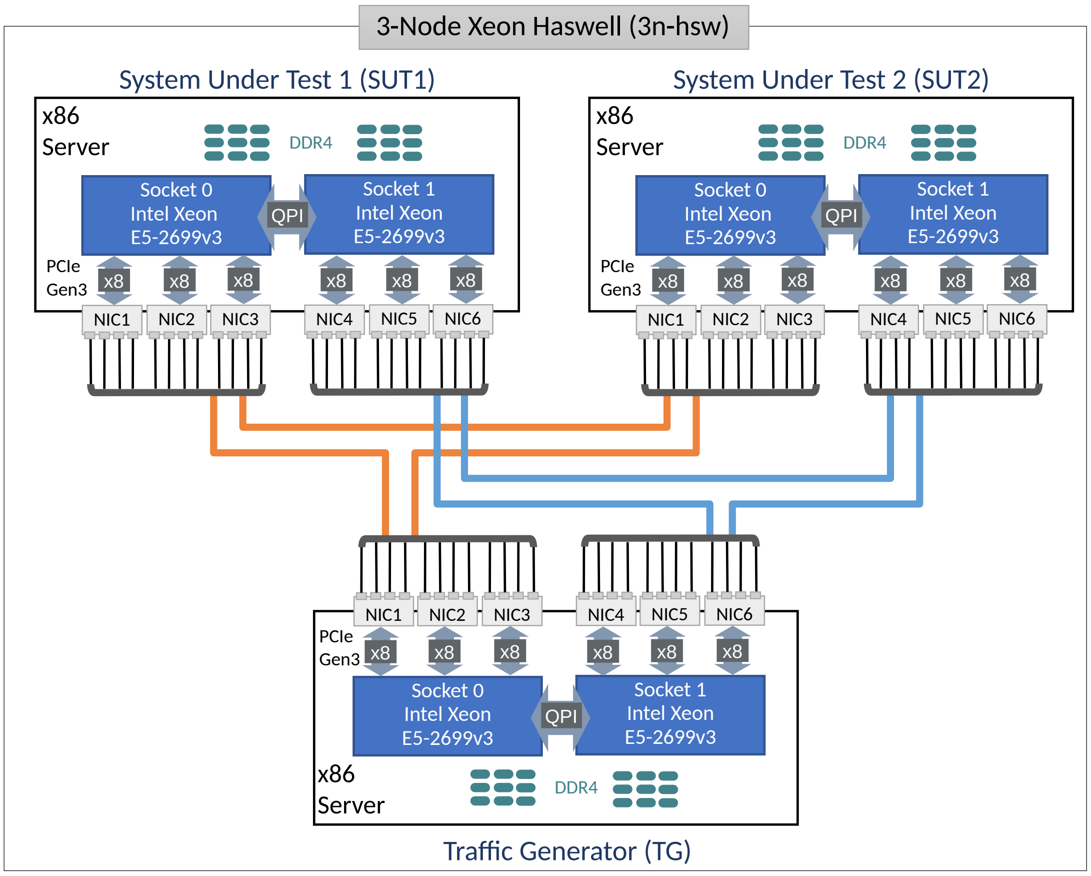
TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.
3. NIC-3: empty, future expansion.
4. NIC-4: empty, future expansion.
5. NIC-5: empty, future expansion.
6. NIC-6: x710-DA4 4p10GE Intel. (For self-tests.)

All Intel Xeon Skylake servers run with Intel Hyper-Threading enabled, doubling the number of logical cores exposed to Linux, with 56 logical cores and 28 physical cores per processor socket.

1.4.4 3-Node Xeon Haswell (3n-hsw)

Three 3n-hsw testbeds are in operation in FD.io labs. Each 3n-hsw testbed is built with three Cisco UCS-c240m3 servers, each in turn equipped with two Intel Xeon Haswell-SP E5-2699v3 processors (45 MB Cache, 2.3 GHz, 18 cores). 3n-hsw physical topology is shown below.


3-Node Xeon Haswell (3n-hsw)
System Under Test 1 (SUT1)
System Under Test 2 (SUT2)
x86 Server
DDR4
Socket 0 Intel Xeon E5-2699v3
Socket 1 Intel Xeon E5-2699v3
QPI
PCIe Gen3
x8
NIC1
NIC2
NIC3
NIC4
NIC5
NIC6
Traffic Generator (TG)

SUT1 and SUT2 servers are populated with the following NIC models:

1. NIC-1: VIC 1385 2p40GE Cisco.
2. NIC-2: NIC x520 2p10GE Intel.
3. NIC-3: empty.

4. NIC-4: NIC xl710-QDA2 2p40GE Intel.
5. NIC-5: NIC x710-DA2 2p10GE Intel.
6. NIC-6: QAT 8950 50G (Walnut Hill) Intel.

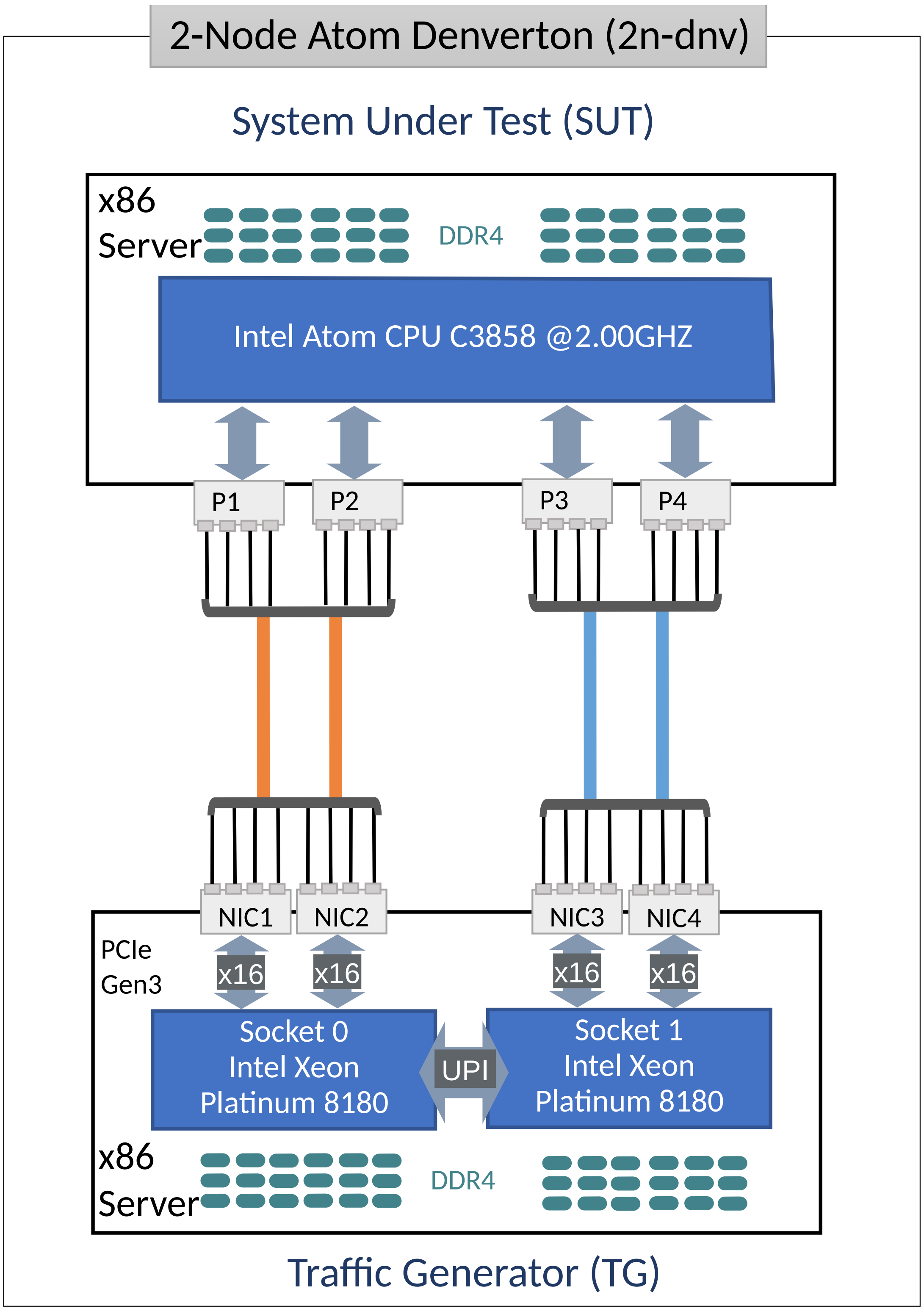
TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: NIC xl710-QDA2 2p40GE Intel.
2. NIC-2: NIC x710-DA2 2p10GE Intel.
3. NIC-3: empty.
4. NIC-4: NIC xl710-QDA2 2p40GE Intel.
5. NIC-5: NIC x710-DA2 2p10GE Intel.
6. NIC-6: NIC x710-DA2 2p10GE Intel. (For self-tests.)

All Intel Xeon Haswell servers run with Intel Hyper-Threading disabled, making the number of logical cores exposed to Linux match the number of 18 physical cores per processor socket.

1.4.5 2-Node Atom Denverton (2n-dnv)

2n-dnv testbed is built with: i) one Intel S2600WFT server acting as TG and equipped with two Intel Xeon Skylake Platinum 8180 processors (38.5 MB Cache, 2.50 GHz, 28 cores), and ii) one SuperMicro SYS-E300-9A server acting as SUT and equipped with one Intel Atom C3858 processor (12 MB Cache, 2.00 GHz, 12 cores). 2n-dnv physical topology is shown below.


2-Node Atom Denverton (2n-dnv)
System Under Test (SUT)
x86 Server
DDR4
Intel Atom CPU C3858 @2.00GHZ
P1
P2
P3
P4
NIC1
NIC2
NIC3
NIC4
PCIe Gen3
x16
x16
x16
x16
Socket 0 Intel Xeon Platinum 8180
UPI
Socket 1 Intel Xeon Platinum 8180
x86 Server
DDR4
Traffic Generator (TG)

SUT server have four internal 10G NIC port:

1. P-1: x553 copper port.
2. P-2: x553 copper port.
3. P-3: x553 fiber port.
4. P-4: x553 fiber port.

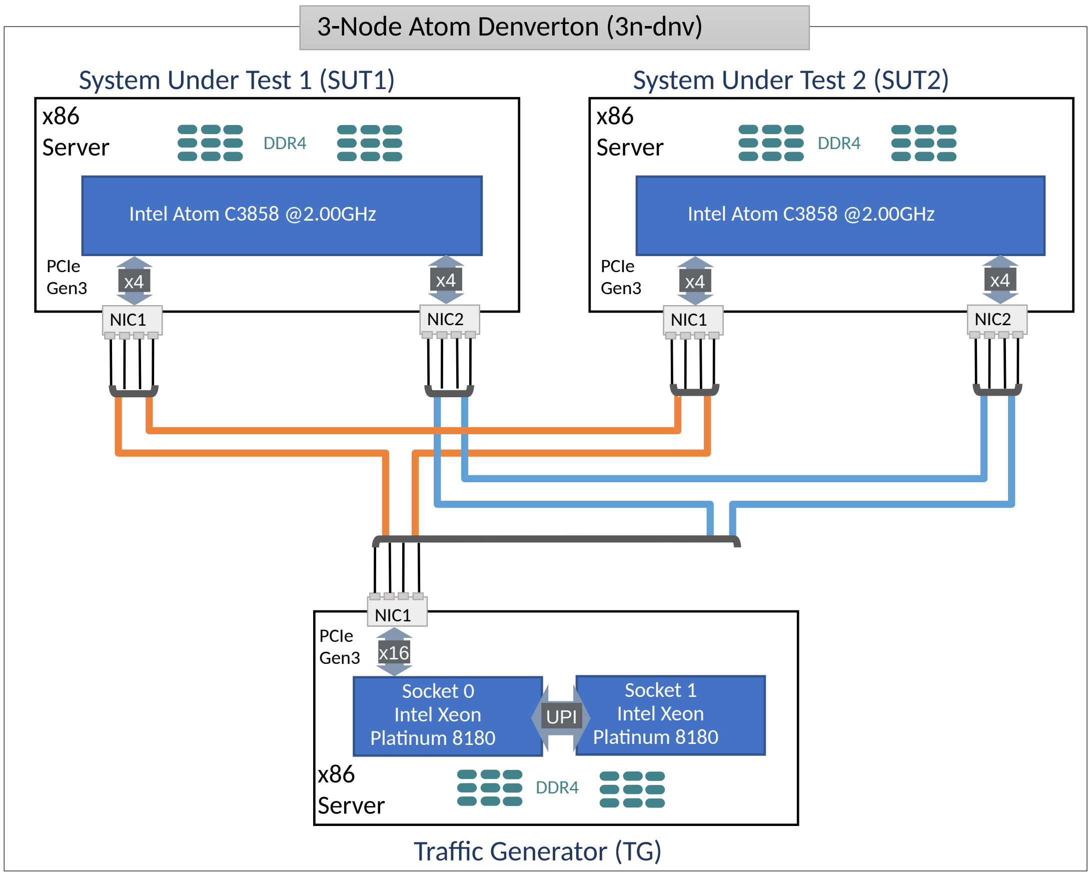
TG server run T-Rex software traffic generator and are populated with the following NIC models:

1. NIC-1: x550-T2 2p10GE Intel.
2. NIC-2: x550-T2 2p10GE Intel.
3. NIC-3: x520-DA2 2p10GE Intel.
4. NIC-4: x520-DA2 2p10GE Intel.

The 2n-dnv testbed is in operation in Intel SH labs.

1.4.6 3-Node Atom Denverton (3n-dnv)

One 3n-dnv testbed is built with: i) one SuperMicro SYS-7049GP-TRT server acting as TG and equipped with two Intel Xeon Skylake Platinum 8180 processors (38.5 MB Cache, 2.50 GHz, 28 cores), and ii) one SuperMicro SYS-E300-9A server acting as SUT and equipped with one Intel Atom C3858 processor (12 MB Cache, 2.00 GHz, 12 cores). 3n-dnv physical topology is shown below.


3-Node Atom Denverton (3n-dnv)
System Under Test 1 (SUT1)
System Under Test 2 (SUT2)
x86 Server
DDR4
Intel Atom C3858 @2.00GHz
PCIe Gen3
x4
NIC1
NIC2
x16
Socket 0 Intel Xeon Platinum 8180
UPI
Socket 1 Intel Xeon Platinum 8180
Traffic Generator (TG)

SUT1 and SUT2 servers are populated with the following NIC models:

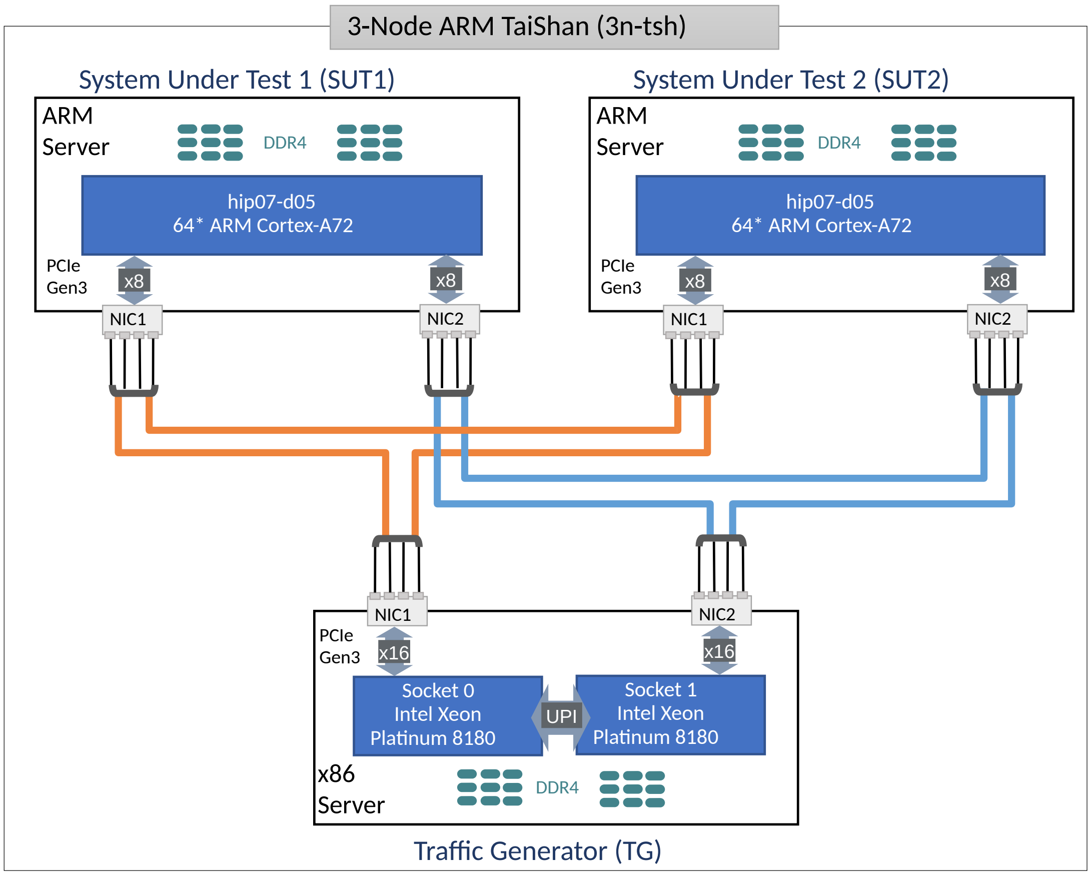
1. NIC-1: x553 2p10GE fiber Intel.
2. NIC-2: x553 2p10GE copper Intel.

TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.

1.4.7 3-Node ARM TaiShan (3n-tsh)

One 3n-tsh testbed is built with: i) one SuperMicro SYS-7049GP-TRT server acting as TG and equipped with two Intel Xeon Skylake Platinum 8180 processors (38.5 MB Cache, 2.50 GHz, 28 cores), and ii) one Huawei TaiShan 2280 server acting as SUT and equipped with one hip07-d05 processor (64* ARM Cortex-A72). 3n-tsh physical topology is shown below.



SUT1 and SUT2 servers are populated with the following NIC models:

1. NIC-1: connectx4 2p25GE Mellanox.
2. NIC-2: x520 2p10GE Intel.

TG servers run T-Rex application and are populated with the following NIC models:

1. NIC-1: x710-DA4 4p10GE Intel.
2. NIC-2: xxv710-DA2 2p25GE Intel.

1.5 Test Methodology

1.5.1 Terminology

- **Frame size**: size of an Ethernet Layer-2 frame on the wire, including any VLAN tags (dot1q, dot1ad) and Ethernet FCS, but excluding Ethernet preamble and inter-frame gap. Measured in Bytes.
- **Packet size**: same as frame size, both terms used interchangeably.
- **Inner L2 size**: for tunneled L2 frames only, size of an encapsulated Ethernet Layer-2 frame, preceded with tunnel header, and followed by tunnel trailer. Measured in Bytes.
- **Inner IP size**: for tunneled IP packets only, size of an encapsulated IPv4 or IPv6 packet, preceded with tunnel header, and followed by tunnel trailer. Measured in Bytes.
- **Device Under Test (DUT)**: In software networking, "device" denotes a specific piece of software tasked with packet processing. Such device is surrounded with other software components (such

as operating system kernel). It is not possible to run devices without also running the other components, and hardware resources are shared between both. For purposes of testing, the whole set of hardware and software components is called "System Under Test" (SUT). As SUT is the part of the whole test setup performance of which can be measured with RFC 2544[2], using SUT instead of RFC 2544[3] DUT. Device under test (DUT) can be re-introduced when analyzing test results using whitebox techniques, but this document sticks to blackbox testing.

- **System Under Test (SUT)**: System under test (SUT) is a part of the whole test setup whose performance is to be benchmarked. The complete methodology contains other parts, whose performance is either already established, or not affecting the benchmarking result.
- **Bi-directional throughput tests**: involve packets/frames flowing in both east-west and west-east directions over every tested interface of SUT/DUT. Packet flow metrics are measured per direction, and can be reported as aggregate for both directions (i.e. throughput) and/or separately for each measured direction (i.e. latency). In most cases bi-directional tests use the same (symmetric) load in both directions.
- **Uni-directional throughput tests**: involve packets/frames flowing in only one direction, i.e. either east-west or west-east direction, over every tested interface of SUT/DUT. Packet flow metrics are measured and are reported for measured direction.
- **Packet Loss Ratio (PLR)**: ratio of packets received relative to packets transmitted over the test trial duration, calculated using formula: PLR = (pkts_transmitted - pkts_received) / pkts_transmitted. For bi-directional throughput tests aggregate PLR is calculated based on the aggregate number of packets transmitted and received.
- **Packet Throughput Rate**: maximum packet offered load DUT/SUT forwards within the specified Packet Loss Ratio (PLR). In many cases the rate depends on the frame size processed by DUT/SUT. Hence packet throughput rate MUST be quoted with specific frame size as received by DUT/SUT during the measurement. For bi-directional tests, packet throughput rate should be reported as aggregate for both directions. Measured in packets-per-second (pps) or frames-per-second (fps), equivalent metrics.
- **Bandwidth Throughput Rate**: a secondary metric calculated from packet throughput rate using formula: bw_rate = pkt_rate * (frame_size + L1_overhead) * 8, where L1_overhead for Ethernet includes preamble (8 Bytes) and inter-frame gap (12 Bytes). For bi-directional tests, bandwidth throughput rate should be reported as aggregate for both directions. Expressed in bits-per-second (bps).
- **Non Drop Rate (NDR)**: maximum packet/bandwith throughput rate sustained by DUT/SUT at PLR equal zero (zero packet loss) specific to tested frame size(s). MUST be quoted with specific packet size as received by DUT/SUT during the measurement. Packet NDR measured in packets-per-second (or fps), bandwidth NDR expressed in bits-per-second (bps).
- **Partial Drop Rate (PDR)**: maximum packet/bandwith throughput rate sustained by DUT/SUT at PLR greater than zero (non-zero packet loss) specific to tested frame size(s). MUST be quoted with specific packet size as received by DUT/SUT during the measurement. Packet PDR measured in packets-per-second (or fps), bandwidth PDR expressed in bits-per-second (bps).
- **Maximum Receive Rate (MRR)**: packet/bandwidth rate regardless of PLR sustained by DUT/SUT under specified Maximum Transmit Rate (MTR) packet load offered by traffic generator. MUST be quoted with both specific packet size and MTR as received by DUT/SUT during the measurement. Packet MRR measured in packets-per-second (or fps), bandwidth MRR expressed in bits-per-second (bps).
- **Trial**: a single measurement step.
- **Trial duration**: amount of time over which packets are transmitted and received in a single measurement step.

[2] https://tools.ietf.org/html/rfc2544.html
[3] https://tools.ietf.org/html/rfc2544.html

1.5.2 VPP Forwarding Modes

VPP is tested in a number of L2 and IP packet lookup and forwarding modes. Within each mode baseline and scale tests are executed, the latter with varying number of lookup entries.

L2 Ethernet Switching

VPP is tested in three L2 forwarding modes:

- *l2patch*: L2 patch, the fastest point-to-point L2 path that loops packets between two interfaces without any Ethernet frame checks or lookups.
- *l2xc*: L2 cross-connect, point-to-point L2 path with all Ethernet frame checks, but no MAC learning and no MAC lookup.
- *l2bd*: L2 bridge-domain, multipoint-to-multipoint L2 path with all Ethernet frame checks, with MAC learning (unless static MACs are used) and MAC lookup.

l2bd tests are executed in baseline and scale configurations:

- *l2bdbase*: low number of L2 flows (254 per direction) is switched by VPP. They drive the content of MAC FIB size (508 total MAC entries). Both source and destination MAC addresses are incremented on a packet by packet basis.
- *l2bdscale*: high number of L2 flows is switched by VPP. Tested MAC FIB sizes include: i) 10k (5k unique flows per direction), ii) 100k (2x 50k flows) and iii) 1M (2x 500k). Both source and destination MAC addresses are incremented on a packet by packet basis, ensuring new entries are learn refreshed and looked up at every packet, making it the worst case scenario.

Ethernet wire encapsulations tested include: untagged, dot1q, dot1ad.

IPv4 Routing

IPv4 routing tests are executed in baseline and scale configurations:

- *ip4base*: low number of IPv4 flows (253 or 254 per direction) is routed by VPP. They drive the content of IPv4 FIB size (506 or 508 total /32 prefixes). Destination IPv4 addresses are incremented on a packet by packet basis.
- *ip4scale*: high number of IPv4 flows is routed by VPP. Tested IPv4 FIB sizes of /32 prefixes include: i) 20k (10k unique flows per direction), ii) 200k (2x 100k flows) and iii) 2M (2x 1M). Destination IPv4 addresses are incremented on a packet by packet basis, ensuring new FIB entries are looked up at every packet, making it the worst case scenario.

IPv6 Routing

IPv6 routing tests are executed in baseline and scale configurations:

- *ip6base*: low number of IPv6 flows (253 or 254 per direction) is routed by VPP. They drive the content of IPv6 FIB size (506 or 508 total /128 prefixes). Destination IPv6 addresses are incremented on a packet by packet basis.
- *ip6scale*: high number of IPv6 flows is routed by VPP. Tested IPv6 FIB sizes of /128 prefixes include: i) 20k (10k unique flows per direction), ii) 200k (2x 100k flows) and iii) 2M (2x 1M). Destination IPv6 addresses are incremented on a packet by packet basis, ensuring new FIB entries are looked up at every packet, making it the worst case scenario.

SRv6 Routing

SRv6 routing tests are executed in a number of baseline configurations, in each case SR policy and steering policy are configured for one direction and one (or two) SR behaviours (functions) in the other directions:

- *srv6enc1sid*: One SID (no SRH present), one SR function - End.
- *srv6enc2sids*: Two SIDs (SRH present), two SR functions - End and End.DX6.
- *srv6enc2sids-nodecaps*: Two SIDs (SRH present) without decapsulation, one SR function - End.
- *srv6proxy-dyn*: Dynamic SRv6 proxy, one SR function - End.AD.
- *srv6proxy-masq*: Masquerading SRv6 proxy, one SR function - End.AM.
- *srv6proxy-stat*: Static SRv6 proxy, one SR function - End.AS.

In all listed cases low number of IPv6 flows (253 per direction) is routed by VPP.

1.5.3 Tunnel Encapsulations

Tunnel encapsulations testing is grouped based on the type of outer header: IPv4 or IPv6.

IPv4 Tunnels

VPP is tested in the following IPv4 tunnel baseline configurations:

- *ip4vxlan-l2bdbase*: VXLAN over IPv4 tunnels with L2 bridge-domain MAC switching.
- *ip4vxlan-l2xcbase*: VXLAN over IPv4 tunnels with L2 cross-connect.
- *ip4lispip4-ip4base*: LISP over IPv4 tunnels with IPv4 routing.
- *ip4lispip6-ip6base*: LISP over IPv4 tunnels with IPv6 routing.

In all cases listed above low number of MAC, IPv4, IPv6 flows (253 or 254 per direction) is switched or routed by VPP.

In addition selected IPv4 tunnels are tested at scale:

- *dot1q–ip4vxlanscale-l2bd*: VXLAN over IPv4 tunnels with L2 bridge- domain MAC switching, with scaled up dot1q VLANs (10, 100, 1k), mapped to scaled up L2 bridge-domains (10, 100, 1k), that are in turn mapped to (10, 100, 1k) VXLAN tunnels. 64.5k flows are transmitted per direction.

IPv6 Tunnels

VPP is tested in the following IPv6 tunnel baseline configurations:

- *ip6lispip4-ip4base*: LISP over IPv4 tunnels with IPv4 routing.
- *ip6lispip6-ip6base*: LISP over IPv4 tunnels with IPv6 routing.

In all cases listed above low number of IPv4, IPv6 flows (253 or 254 per direction) is routed by VPP.

1.5.4 VPP Features

VPP is tested in a number of data plane feature configurations across different forwarding modes. Following sections list features tested.

ACL Security-Groups

Both stateless and stateful access control lists (ACL), also known as security-groups, are supported by VPP.

Following ACL configurations are tested for MAC switching with L2 bridge-domains:

- *l2bdbasemaclrn-iacl{E}sl-{F}flows*: Input stateless ACL, with {E} entries and {F} flows.
- *l2bdbasemaclrn-oacl{E}sl-{F}flows*: Output stateless ACL, with {E} entries and {F} flows.
- *l2bdbasemaclrn-iacl{E}sf-{F}flows*: Input stateful ACL, with {E} entries and {F} flows.
- *l2bdbasemaclrn-oacl{E}sf-{F}flows*: Output stateful ACL, with {E} entries and {F} flows.

Following ACL configurations are tested with IPv4 routing:

- *ip4base-iacl{E}sl-{F}flows*: Input stateless ACL, with {E} entries and {F} flows.
- *ip4base-oacl{E}sl-{F}flows*: Output stateless ACL, with {E} entries and {F} flows.
- *ip4base-iacl{E}sf-{F}flows*: Input stateful ACL, with {E} entries and {F} flows.
- *ip4base-oacl{E}sf-{F}flows*: Output stateful ACL, with {E} entries and {F} flows.

ACL tests are executed with the following combinations of ACL entries and number of flows:

- ACL entry definitions
  - flow non-matching deny entry: (src-ip4, dst-ip4, src-port, dst-port).
  - flow matching permit ACL entry: (src-ip4, dst-ip4).
- {E} - number of non-matching deny ACL entries, {E} = [1, 10, 50].
- {F} - number of UDP flows with different tuple (src-ip4, dst-ip4, src-port, dst-port), {F} = [100, 10k, 100k].
- All {E}x{F} combinations are tested per ACL type, total of 9.

ACL MAC-IP

MAC-IP binding ACLs are tested for MAC switching with L2 bridge-domains:

- *l2bdbasemaclrn-macip-iacl{E}sl-{F}flows*: Input stateless ACL, with {E} entries and {F} flows.

MAC-IP ACL tests are executed with the following combinations of ACL entries and number of flows:

- ACL entry definitions
  - flow non-matching deny entry: (dst-ip4, dst-mac, bit-mask)
  - flow matching permit ACL entry: (dst-ip4, dst-mac, bit-mask)
- {E} - number of non-matching deny ACL entries, {E} = [1, 10, 50]
- {F} - number of UDP flows with different tuple (dst-ip4, dst-mac), {F} = [100, 10k, 100k]
- All {E}x{F} combinations are tested per ACL type, total of 9.

NAT44

NAT44 is tested in baseline and scale configurations with IPv4 routing:

- *ip4base-nat44*: baseline test with single NAT entry (addr, port), single UDP flow.
- *ip4base-udpsrcscale{U}-nat44*: baseline test with {U} NAT entries (addr, {U}ports), {U}=15.
- *ip4scale{R}-udpsrcscale{U}-nat44*: scale tests with {R}*{U} NAT entries ({R}addr, {U}ports), {R}=[100, 1k, 2k, 4k], {U}=15.

1.5.5 Data Plane Throughput

Data Plane Throughput Tests

Network data plane throughput is measured using multiple test methods in order to obtain representative and repeatable results across the large set of performance test cases implemented and executed within CSIT.

Following throughput test methods are used:

- MLRsearch - Multiple Loss Ratio search
- MRR - Maximum Receive Rate
- PLRsearch - Probabilistic Loss Ratio search

Description of each test method is followed by generic test properties shared by all methods.

MLRsearch Tests

Description

Multiple Loss Ratio search (MLRsearch) tests discover multiple packet throughput rates in a single search, reducing the overall test execution time compared to a binary search. Each rate is associated with a distinct Packet Loss Ratio (PLR) criteria. In FD.io CSIT two throughput rates are discovered: Non-Drop Rate (NDR, with zero packet loss, PLR=0) and Partial Drop Rate (PDR, with PLR<0.5%). MLRsearch is compliant with **RFC 2544**[4].

Usage

MLRsearch tests are run to discover NDR and PDR rates for each VPP and DPDK release covered by CSIT report. Results for small frame sizes (64b/78B, IMIX) are presented in packet throughput graphs (Box-and-Whisker Plots) with NDR and PDR rates plotted against the test cases covering popular VPP packet paths.

Each test is executed at least 10 times to verify measurements repeatability and results are compared between releases and test environments. NDR and PDR packet and bandwidth throughput results for all frame sizes and for all tests are presented in detailed results tables.

Details

See *MLRsearch Tests* (page 20) section for more detail. MLRsearch is being standardized in IETF in draft-vpolak-mkonstan-mlrsearch[5].

MRR Tests

Description

Maximum Receive Rate (MRR) tests are complementary to MLRsearch tests, as they provide a maximum “raw” throughput benchmark for development and testing community.

MRR tests measure the packet forwarding rate under the maximum load offered by traffic generator (dependent on link type and NIC model) over a set trial duration, regardless of packet loss. Maximum load for specified Ethernet frame size is set to the bi-directional link rate.

[4] https://tools.ietf.org/html/rfc2544.html
[5] https://tools.ietf.org/html/draft-vpolak-mkonstan-bmwg-mlrsearch

Usage

MRR tests are much faster than MLRsearch as they rely on a single trial or a small set of trials with very short duration. It is this property that makes them suitable for continuous execution in daily performance trending jobs enabling detection of performance anomalies (regressions, progressions) resulting from data plane code changes.

MRR tests are also used for VPP per patch performance jobs verifying patch performance vs. parent. CSIT reports include MRR throughput comparisons between releases and test environments. Small frame sizes only (64b/78B, IMIX).

Details

See *MRR Throughput* (page 20) section for more detail about MRR tests configuration.

FD.io CSIT performance dashboard includes complete description of daily performance trending tests[6] and VPP per patch tests[7].

PLRsearch Tests

Description

Probabilistic Loss Ratio search (PLRsearch) tests discovers a packet throughput rate associated with configured Packet Loss Ratio (PLR) criteria for tests run over an extended period of time a.k.a. soak testing. PLRsearch assumes that system under test is probabilistic in nature, and not deterministic.

Usage

PLRsearch are run to discover a sustained throughput for PLR=10^-7 (close to NDR) for VPP release covered by CSIT report. Results for small frame sizes (64b/78B) are presented in packet throughput graphs (Box Plots) for a small subset of baseline tests.

Each soak test lasts 30 minutes and is executed at least twice. Results are compared against NDR and PDR rates discovered with MLRsearch.

Details

See *PLRsearch* (page 21) methodology section for more detail. PLRsearch is being standardized in IETF in draft-vpolak-bmwg-plrsearch[8].

Generic Test Properties

All data plane throughput test methodologies share following generic properties:

- Tested L2 frame sizes (untagged Ethernet):
  - IPv4 payload: 64B, IMIX (28x64B, 16x570B, 4x1518B), 1518B, 9000B.
  - IPv6 payload: 78B, IMIX (28x78B, 16x570B, 4x1518B), 1518B, 9000B.
  - All quoted sizes include frame CRC, but exclude per frame transmission overhead of 20B (preamble, inter frame gap).

[6] https://docs.fd.io/csit/master/trending/methodology/performance_tests.html
[7] https://docs.fd.io/csit/master/trending/methodology/perpatch_performance_tests.html
[8] https://tools.ietf.org/html/draft-vpolak-bmwg-plrsearch

- Offered packet load is always bi-directional and symmetric.
- All measured and reported packet and bandwidth rates are aggregate bi-directional rates reported from external Traffic Generator perspective.

MLRsearch Tests

Overview

Multiple Loss Rate search (MLRsearch) tests use new search algorithm implemented in FD.io CSIT project. MLRsearch discovers multiple packet throughput rates in a single search, with each rate associated with a different Packet Loss Ratio (PLR) criteria.

Two throughput measurements used in FD.io CSIT are Non-Drop Rate (NDR, with zero packet loss, PLR=0) and Partial Drop Rate (PDR, with packet loss rate not greater than the configured non-zero PLR).

MLRsearch discovers NDR and PDR in a single pass reducing required time duration compared to separate ‘binary search‘_es for NDR and PDR. Overall search time is reduced even further by relying on shorter trial durations of intermediate steps, with only the final measurements conducted at the specified final trial duration. This results in the shorter overall execution time when compared to standard NDR/PDR binary search, while guaranteeing similar results.

If needed, next version of MLRsearch can be easily adopted to discover more throughput rates with different pre-defined PLRs.

Note: All throughput rates are *always* bi-directional aggregates of two equal (symmetric) uni-directional packet rates received and reported by an external traffic generator.

Search Implementation

Detailed description of the MLRsearch algorithm is included in the IETF draft draft-vpolak-mkonstan-mlrsearch[9] that is in the process of being standardized in the IETF Benchmarking Methodology Working Group (BMWG).

MLRsearch is also available as a PyPI (Python Package Index) library[10].

Implementation Deviations

FD.io CSIT implementation of MLRsearch so far is fully based on the -02 version of the draft-vpolak-mkonstan-mlrsearch-02[11].

MRR Throughput

Maximum Receive Rate (MRR) tests are complementary to MLRsearch tests, as they provide a maximum “raw” throughput benchmark for development and testing community. MRR tests measure the packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss.

MRR tests are currently used for following test jobs:

- Report performance comparison: 64B, IMIX for vhost, memif.
- Daily performance trending: 64B, IMIX for vhost, memif.

[9] https://tools.ietf.org/html/draft-vpolak-mkonstan-bmwg-mlrsearch
[10] https://pypi.org/project/MLRsearch/
[11] https://tools.ietf.org/html/draft-vpolak-mkonstan-bmwg-mlrsearch-02

- Per-patch performance verification: 64B.
- Initial iterations of MLRsearch and PLRsearch: 64B.

Maximum offered load for specific L2 Ethernet frame size is set to either the maximum bi-directional link rate or tested NIC model capacity, as follows:

- For 10GE NICs the maximum packet rate load is 2x14.88 Mpps for 64B, a 10GE bi-directional link rate.
- For 25GE NICs the maximum packet rate load is 2x18.75 Mpps for 64B, a 25GE bi-directional link sub-rate limited by 25GE NIC used on TRex TG, XXV710.
- For 40GE NICs the maximum packet rate load is 2x18.75 Mpps for 64B, a 40GE bi-directional link sub-rate limited by 40GE NIC used on TRex TG,XL710. Packet rate for other tested frame sizes is limited by PCIeGen3 x8 bandwidth limitation of ~50Gbps.

MRR test code implements multiple bursts of offered packet load and has two configurable burst parameters: individual trial duration and number of trials in a single burst. This enables more precise performance trending by providing more results data for analysis.

Burst parameter settings vary between different tests using MRR:

- MRR individual trial duration:
  - Report performance comparison: 1 sec.
  - Daily performance trending: 1 sec.
  - Per-patch performance verification: 10 sec.
  - Initial iteration for MLRsearch: 1 sec.
  - Initial iteration for PLRsearch: 5.2 sec.
- Number of MRR trials per burst:
  - Report performance comparison: 10.
  - Daily performance trending: 10.
  - Per-patch performance verification: 5.
  - Initial iteration for MLRsearch: 1.
  - Initial iteration for PLRsearch: 1.

PLRsearch

Motivation for PLRsearch

Network providers are interested in throughput a system can sustain.

RFC 2544[12] assumes loss ratio is given by a deterministic function of offered load. But NFV software systems are not deterministic enough. This makes deterministic algorithms (such as binary search[13] per RFC 2544 and MLRsearch with single trial) to return results, which when repeated show relatively high standard deviation, thus making it harder to tell what "the throughput" actually is.

We need another algorithm, which takes this indeterminism into account.

Generic Algorithm

Detailed description of the PLRsearch algorithm is included in the IETF draft draft-vpolak-bmwg-plrsearch-02[14] that is in the process of being standardized in the IETF Benchmarking Methodology Work-

[12] https://tools.ietf.org/html/rfc2544
[13] https://en.wikipedia.org/wiki/Binary_search_algorithm
[14] https://tools.ietf.org/html/draft-vpolak-bmwg-plrsearch-02

ing Group (BMWG).

Terms

The rest of this page assumes the reader is familiar with the following terms defined in the IETF draft:

- Trial Order Independent System
- Duration Independent System
- Target Loss Ratio
- Critical Load
- Offered Load regions
  - Zero Loss Region
  - Non-Deterministic Region
  - Guaranteed Loss Region
- Fitting Function
  - Stretch Function
  - Erf Function
- Bayesian Inference
  - Prior distribution
  - Posterior Distribution
- Numeric Integration
  - Monte Carlo
  - Importance Sampling

FD.io CSIT Implementation Specifics

The search receives min_rate and max_rate values, to avoid measurements at offered loads not supporeted by the traffic generator.

The implemented tests cases use bidirectional traffic. The algorithm stores each rate as bidirectional rate (internally, the algorithm is agnostic to flows and directions, it only cares about aggregate counts of packets sent and packets lost), but debug output from traffic generator lists unidirectional values.

In a sample implemenation in FD.io CSIT project, there is roughly 0.5 second delay between trials due to restrictons imposed by packet traffic generator in use (T-Rex).

As measurements results come in, posterior distribution computation takes more time (per sample), although there is a considerable constant part (mostly for inverting the fitting functions).

Also, the integrator needs a fair amount of samples to reach the region the posterior distribution is concentrated at.

And of course, the speed of the integrator depends on computing power of the CPU the algorithm is able to use.

All those timing related effects are addressed by arithmetically increasing trial durations with configurable coefficients (currently 5.1 seconds for the first trial, each subsequent trial being 0.1 second longer).

In order to avoid them, the current implementation tracks natural logarithm (instead of the original quantity) for any quantity which is never negative. Logarithm of zero is minus infinity (not supported by Python), so special value "None" is used instead. Specific functions for frequent operations (such as "logarithm of sum of exponentials") are defined to handle None correctly.

Current implementation uses two fitting functions, called “stretch” and “erf”. In general, their estimates for critical rate differ, which adds a simple source of systematic error, on top of randomness error reported by integrator. Otherwise the reported stdev of critical rate estimate is unrealistically low.

Both functions are not only increasing, but also convex (meaning the rate of increase is also increasing).

Both fitting functions have several mathematically equivalent formulas, each can lead to an arithmetic overflow or underflow in different sub-terms. Overflows can be eliminated by using different exact formulas for different argument ranges. Underflows can be avoided by using approximate formulas in affected argument ranges, such ranges have their own formulas to compute. At the end, both fitting function implementations contain multiple “if” branches, discontinuities are a possibility at range boundaries.

The numeric integrator expects all the parameters to be distributed (independently and) uniformly on an interval (-1, 1).

As both “mrr” and “spread” parameters are positive and not dimensionless, a transformation is needed. Dimentionality is inherited from max_rate value.

The “mrr” parameter follows a Lomax distribution[15] with alpha equal to one, but shifted so that mrr is always greater than 1 packet per second.

The “stretch” parameter is generated simply as the “mrr” value raised to a random power between zero and one; thus it follows a reciprocal distribution[16].

After few measurements, the posterior distribution of fitting function arguments gets quite concentrated into a small area. The integrator is using Monte Carlo[17] with importance sampling[18] where the biased distribution is bivariate Gaussian[19] distribution, with deliberately larger variance. If the generated sample falls outside (-1, 1) interval, another sample is generated.

The center and the covariance matrix for the biased distribution is based on the first and second moments of samples seen so far (within the computation). The center is used directly, covariance matrix is scaled up by a heurictic constant (8.0 by default). The following additional features are applied designed to avoid hyper-focused distributions.

Each computation starts with the biased distribution inherited from the previous computation (zero point and unit covariance matrix is used in the first computation), but the overal weight of the data is set to the weight of the first sample of the computation. Also, the center is set to the first sample point. When additional samples come, their weight (including the importance correction) is compared to sum of the weights of data seen so far (within the iteration). If the new sample is more than one e-fold more impactful, both weight values (for data so far and for the new sample) are set to (geometric) average of the two weights.

This combination showed the best behavior, as the integrator usually follows two phases. First phase (where inherited biased distribution or single big sample are dominating) is mainly important for locating the new area the posterior distribution is concentrated at. The second phase (dominated by whole sample population) is actually relevant for the critical rate estimation.

First two measurements are hardcoded to happen at the middle of rate interval and at max_rate. Next two measurements follow MRR-like logic, offered load is decreased so that it would reach target loss ratio if offered load decrease lead to equal decrease of loss rate.

The rest of measurements start directly in between erf and stretch estimate average. There is one workaround implemented, aimed at reducing the number of consequent zero loss measurements (per fitting function). The workaround first stores every measurement result which loss ratio was the targed loss ratio or higher. Sorted list (called lossy loads) of such results is maintained.

When a sequence of one or more zero loss measurement results is encountered, a smallest of lossy loads is drained from the list. If the estimate average is smaller than the drained value, a weighted average of this estimate and the drained value is used as the next offered load. The weight of the estimate decreases exponentially with the length of consecutive zero loss results.

[15] https://en.wikipedia.org/wiki/Lomax_distribution
[16] https://en.wikipedia.org/wiki/Reciprocal_distribution
[17] https://en.wikipedia.org/wiki/Monte_Carlo_integration
[18] https://en.wikipedia.org/wiki/Importance_sampling
[19] https://en.wikipedia.org/wiki/Multivariate_normal_distribution

This behavior helps the algorithm with convergence speed, as it does not need so many zero loss result to get near critical region. Using the smallest (not drained yet) of lossy loads makes it sure the new offered load is unlikely to result in big loss region. Draining even if the estimate is large enough helps to discard early measurements when loss hapened at too low offered load. Current implementation adds 4 copies of lossy loads and drains 3 of them, which leads to fairly stable behavior even for somewhat inconsistent SUTs.

As high loss count measurements add many bits of information, they need a large amount of small loss count measurements to balance them, making the algorithm converge quite slowly. Typically, this happens when few initial measurements suggest spread way bigger then later measurements. The workaround in offered load selection helps, but more intelligent workarounds could get faster convergence still.

Some systems evidently do not follow the assumption of repeated measurements having the same average loss rate (when the offered load is the same). The idea of estimating the trend is not implemented at all, as the observed trends have varied characteristics.

Probably, using a more realistic fitting functions will give better estimates than trend analysis.

Bottom Line

The notion of Throughput is easy to grasp, but it is harder to measure with any accuracy for non-deterministic systems.

Even though the notion of critical rate is harder to grasp than the notion of throughput, it is easier to measure using probabilistic methods.

In testing, the difference between througput measurements and critical rate measurements is usually small, see soak vs ndr comparison.

In pactice, rules of thumb such as “send at max 95% of purported throughput” are common. The correct benchmarking analysis should ask “Which notion is 95% of throughput an approximation to?” before attempting to answer “Is 95% of critical rate safe enough?”.

Algorithmic Analysis

While the estimation computation is based on hard probability science; the offered load selection part of PLRsearch logic is pure heuristics, motivated by what would a human do based on measurement and computation results.

The quality of any heuristic is not affected by soundness of its motivation, just by its ability to achieve the intended goals. In case of offered load selection, the goal is to help the search to converge to the long duration estimates sooner.

But even those long duration estimates could still be of poor quality. Even though the estimate computation is Bayesian (so it is the best it could be within the applied assumptions), it can still of poor quality when compared to what a human would estimate.

One possible source of poor quality is the randomnes inherently present in Monte Carlo numeric integration, but that can be supressed by tweaking the time related input parameters.

The most likely source of poor quality then are the assumptions. Most importantly, the number and the shape of fitting functions; but also others, such as trial order independence and duration independence.

The result can have poor quality in basically two ways. One way is related to location. Both upper and lower bounds can be overestimates or underestimates, meaning the entire estimated interval between lower bound and upper bound lays above or below (respectively) of human-estimated interval. The other way is related to the estimation interval width. The interval can be too wide or too narrow, compared to human estimation.

An estimate from a particular fitting function can be classified as an overestimate (or underestimate) just by looking at time evolution (without human examining measurement results). Overestimates decrease by time, underestimates increase by time (assuming the system performance stays constant).

Quality of the width of the estimation interval needs human evaluation, and is unrelated to both rate of narrowing (both good and bad estimate intervals get narrower at approximately the same relative rate) and relatative width (depends heavily on the system being tested).

The following pictures show the upper (red) and lower (blue) bound, as well as average of Stretch (pink) and Erf (light green) estimate, and offered load chosen (grey), as computed by PLRsearch, after each trial measurement within the 30 minute duration of a test run.

Both graphs are focusing on later estimates. Estimates computed from few initial measurements are wildly off the y-axis range shown.

The following analysis will rely on frequency of zero loss measurements and magnitude of loss ratio if nonzero.

The offered load selection strategy used implies zero loss measurements can be gleaned from the graph by looking at offered load points. When the points move up farther from lower estimate, it means the previous measurement had zero loss. After non-zero loss, the offered load starts again right between (the previous values of) the estimate curves.

The very big loss ratio results are visible as noticeable jumps of both estimates downwards. Medium and small loss ratios are much harder to distinguish just by looking at the estimate curves, the analysis is based on raw loss ratio measurement results.

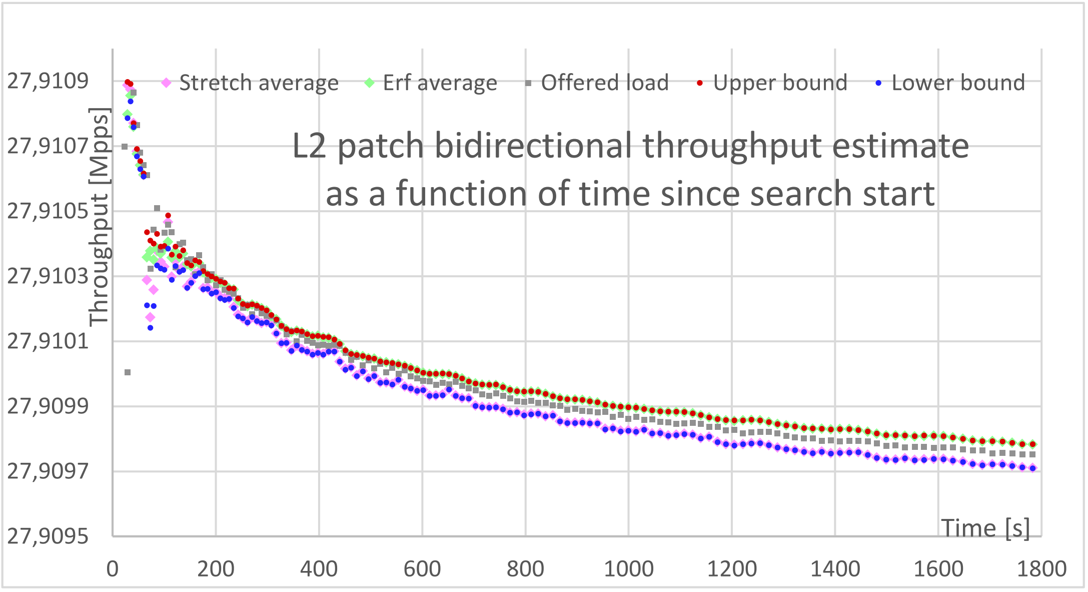
The following descriptions should explain why the graphs seem to signal low quality estimate at first sight, but a more detailed look reveals the quality is good (considering the measurement results).

L2 patch

Both fitting functions give similar estimates, the graph shows "stochasticity" of measurements (estimates increase and decrease within small time regions), and an overall trend of decreasing estimates.

On the first look, the final interval looks fairly narrow, especially compared to the region the estimates have travelled during the search. But the look at the frequency of zero loss results shows this is not a case of overestimation. Measurements at around the same offered load have higher probability of zero loss earlier (when performed farther from upper bound), but smaller probability later (when performed closer to upper bound). That means it is the performance of the system under test that decreases (slightly) over time.

With that in mind, the apparent narrowness of the interval is not a sign of low quality, just a consequence of PLRsearch assuming the performance stays constant.

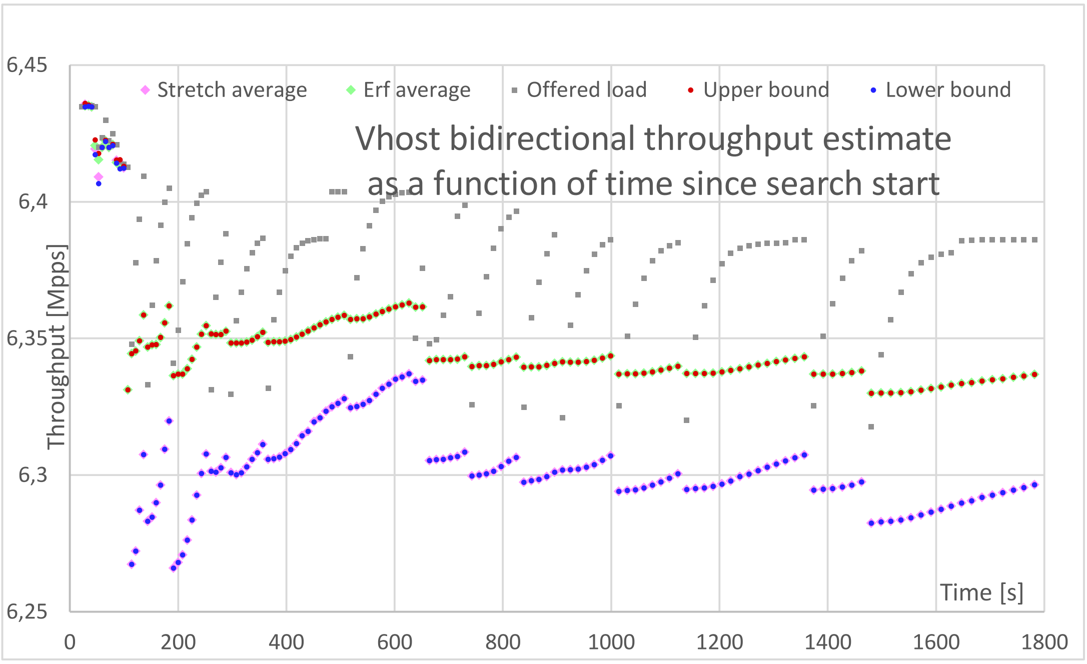


Vhost

This test case shows what looks like a quite broad estimation interval, compared to other test cases with similarly looking zero loss frequencies. Notable features are infrequent high-loss measurement results causing big drops of estimates, and lack of long-term convergence.

Any convergence in medium-sized intervals (during zero loss results) is reverted by the big loss results, as they happen quite far from the critical load estimates, and the two fitting functions extrapolate differently.

In other words, human only seeing estimates from one fitting function would expect narrower end interval, but human seeing the measured loss ratios agrees that the interval should be wider than that.



Summary

The two graphs show the behavior of PLRsearch algorithm applied to soaking test when some of PLRsearch assumptions do not hold:

- L2 patch measurement results violate the assumption of performance not changing over time.
- Vhost measurement results violate the assumption of Poisson distribution matching the loss counts.

The reported upper and lower bounds can have distance larger or smaller than a first look by a human would expect, but a more closer look reveals the quality is good, considering the circumstances.

The usefullness of the critical load estimate is of questionable value when the assumptions are violated.

Some improvements can be made via more specific workarounds, for example long term limit of L2 patch performance could be estmated by some heuristic.

Other improvements can be achieved only by asking users whether loss patterns matter. Is it better to have single digit losses distributed fairly evenly over time (as Poisson distribution would suggest), or is it better to have short periods of medium losses mixed with long periods of zero losses (as happens in Vhost test) with the same overall loss ratio?

1.5.6 Packet Latency

TRex Traffic Generator (TG) is used for measuring latency across 2-Node and 3-Node SUT server topologies. TRex integrates A High Dynamic Range Histogram (HDRH)[20] code providing per packet latency distribution for latency streams sent in parallel to the main load packet streams. Packet latency is measured using following methodology:

- Latency tests are performed at following packet load levels:
  - No-Load: latency streams only.
  - Low-Load: at 10% PDR.
  - Mid-Load: at 50% PDR.
  - High-Load: at 90% PDR.
  - NDR-Load: at 100% NDR.
  - PDR-Load: at 100% PDR.
- Latency is measured for all tested packet sizes except IMIX due to TG restriction.
- TG sends dedicated latency streams, one per direction, each at the rate of 9 kpps at the prescribed packet size; these are sent in addition to the main load streams.
- TG reports Min/Avg/Max and HDRH latency values distribution per stream direction, hence two sets of latency values are reported per test case.
- Reported latency values are aggregate across tested topology.
- +/- 1 usec is the measurement accuracy advertised by TRex TG for the setup used.
- TG setup introduces an always-on Tx/Rx interface latency of about 2 * 2 usec per direction induced by TRex SW writing and reading packet timestamps on CPU cores.

1.5.7 Multi-Core Speedup

All performance tests are executed with single physical core and with multiple cores scenarios.

Intel Hyper-Threading (HT)

Intel Xeon processors used in FD.io CSIT can operate either in HT Disabled mode (single logical core per each physical core) or in HT Enabled mode (two logical cores per each physical core). HT setting is applied in BIOS and requires server SUT reload for it to take effect, making it impractical for continuous changes of HT mode of operation.

CSIT-1908.2 performance tests are executed with server SUTs' Intel XEON processors configured with Intel Hyper-Threading Disabled for all Xeon Haswell testbeds (3n-hsw) and with Intel Hyper-Threading Enabled for all Xeon Skylake and Xeon Cascadelake testbeds.

More information about physical testbeds is provided in *Physical Testbeds* (page 3).

Multi-core Tests

CSIT-1908.2 multi-core tests are executed in the following VPP worker thread and physical core configurations:

1. Intel Xeon Haswell testbeds (3n-hsw) with Intel HT disabled (1 logical CPU core per each physical core):

1. 1t1c - 1 VPP worker thread on 1 physical core.

[20] http://hdrhistogram.org/

2. 2t2c - 2 VPP worker threads on 2 physical cores.
3. 4t4c - 4 VPP worker threads on 4 physical cores.

1. Intel Xeon Skylake and Cascadelake testbeds (2n-skx, 3n-skx, 2n-clx) with Intel HT enabled (2 logical CPU cores per each physical core):

1. 2t1c - 2 VPP worker threads on 1 physical core.
2. 4t2c - 4 VPP worker threads on 2 physical cores.
3. 8t4c - 8 VPP worker threads on 4 physical cores.

VPP worker threads are the data plane threads running on isolated logical cores. With Intel HT enabled VPP workers are placed as sibling threads on each used physical core. VPP control threads (main, stats) are running on a separate non-isolated core together with other Linux processes.

In all CSIT tests care is taken to ensure that each VPP worker handles the same amount of received packet load and does the same amount of packet processing work. This is achieved by evenly distributing per interface type (e.g. physical, virtual) receive queues over VPP workers using default VPP round-robin mapping and by loading these queues with the same amount of packet flows.

If number of VPP workers is higher than number of physical or virtual interfaces, multiple receive queues are configured on each interface. NIC Receive Side Scaling (RSS) for physical interfaces and multi-queue for virtual interfaces are used for this purpose.

Section throughput_speedup_multi_core includes a set of graphs illustrating packet throughout speedup when running VPP worker threads on multiple cores. Note that in quite a few test cases running VPP workers on 2 or 4 physical cores hits the I/O bandwidth or packets-per-second limit of tested NIC.

1.5.8 Hoststack Testing

HTTP/TCP with WRK

WRK HTTP benchmarking tool[21] is used for TCP/IP and HTTP tests of VPP Host Stack and built-in static HTTP server. WRK has been chosen as it is capable of generating significant TCP/IP and HTTP loads by scaling number of threads across multi-core processors.

This in turn enables high scale benchmarking of the VPP Host Stack TCP/IP and HTTP service including HTTP TCP/IP Connections-Per-Second (CPS) and HTTP Requests-Per-Second.

The initial tests are designed as follows:

- HTTP and TCP/IP Connections-Per-Second (CPS)
  - WRK configured to use 8 threads across 8 cores, 1 thread per core.
  - Maximum of 50 concurrent connections across all WRK threads.
  - Timeout for server responses set to 5 seconds.
  - Test duration is 30 seconds.
  - Expected HTTP test sequence:
    - Single HTTP GET Request sent per open connection.
    - Connection close after valid HTTP reply.
    - Resulting flow sequence - 8 packets: >Syn, <Syn-Ack, >Ack, >Req, <Rep, >Fin, <Fin, >Ack.
- HTTP Requests-Per-Second
  - WRK configured to use 8 threads across 8 cores, 1 thread per core.
  - Maximum of 50 concurrent connections across all WRK threads.

[21] https://github.com/wg/wrk

  - Timeout for server responses set to 5 seconds.
  - Test duration is 30 seconds.
  - Expected HTTP test sequence:
    * Multiple HTTP GET Requests sent in sequence per open connection.
    * Connection close after set test duration time.
    * Resulting flow sequence: >Syn, <Syn-Ack, >Ack, >Req[1], <Rep[1], .., >Req[n], <Rep[n], >Fin, <Fin, >Ack.

TCP/IP with iperf3

iperf3 goodput measurement tool[22] is used for measuring the maximum attainable goodput of the VPP Host Stack connection across two instances of VPP running on separate DUT nodes. iperf3 is a popular open source tool for active measurements of the maximum achievable goodput on IP networks.

Because iperf3 utilizes the POSIX socket interface APIs, the current test configuration utilizes the LD_PRELOAD mechanism in the linux kernel to connect iperf3 to the VPP Host Stack using the VPP Communications Library (VCL) LD_PRELOAD library (libvcl_ldpreload.so).

In the future, a forked version of iperf3 which has been modified to directly use the VCL application APIs may be added to determine the difference in performance of 'VCL Native' applications versus utilizing LD_PRELOAD which inherently has more overhead and other limitations.

The test configuration is as follows:

```
       DUT1                 Network                DUT2
[ iperf3-client -> VPP1 ]=======[ VPP2 -> iperf3-server]
```

where,

1. iperf3 server attaches to VPP2 and LISTENs on VPP2:TCP port 5201.
2. iperf3 client attaches to VPP1 and opens one or more stream connections to VPP2:TCP port 5201.
3. iperf3 client transmits a uni-directional stream as fast as the VPP Host Stack allows to the iperf3 server for the test duration.
4. At the end of the test the iperf3 client emits the goodput measurements for all streams and the sum of all streams.

Test cases include 1 and 10 Streams with a 20 second test duration with the VPP Host Stack configured to utilize the Cubic TCP congestion algorithm.

Note: iperf3 is single threaded, so it is expected that the 10 stream test does not show any performance improvement due to multi-thread/multi-core execution.

There are also variations of these test cases which use the VPP Network Simulator (NSIM) plugin to test the VPP Hoststack goodput with 1 percent of the traffic being dropped at the output interface of VPP1 thereby simulating a lossy network. The NSIM tests are experimental and the test results are not currently representative of typical results in a lossy network.

QUIC/UDP/IP with vpp_echo

vpp_echo performance testing tool[23] is a bespoke performance test application which utilizes the 'native HostStack APIs' to verify performance and correct handling of connection/stream events with uni-directional and bi-directional streams of data.

Because iperf3 does not support the QUIC transport protocol, vpp_echo is used for measuring the maximum attainable goodput of the VPP Host Stack connection utilizing the QUIC transport protocol across

[22] https://github.com/esnet/iperf
[23] https://wiki.fd.io/view/VPP/HostStack#External_Echo_Server.2FClient_.28vpp_echo.29

two instances of VPP running on separate DUT nodes. The QUIC transport protocol supports multiple streams per connection and test cases utilize different combinations of QUIC connections and number of streams per connection.

The test configuration is as follows:

```
        DUT1                  Network                  DUT2
[ vpp_echo-client -> VPP1 ]=======[ VPP2 -> vpp_echo-server]
                      N-streams/connection
```

where,

1. vpp_echo server attaches to VPP2 and LISTENs on VPP2:TCP port 1234.
2. vpp_echo client creates one or more connections to VPP1 and opens one or more stream per connection to VPP2:TCP port 1234.
3. vpp_echo client transmits a uni-directional stream as fast as the VPP Host Stack allows to the vpp_echo server for the test duration.
4. At the end of the test the vpp_echo client emits the goodput measurements for all streams and the sum of all streams.

Test cases include

1. 1 QUIC Connection with 1 Stream
2. 1 QUIC connection with 10 Streams
3. 10 QUIC connetions with 1 Stream
4. 10 QUIC connections with 10 Streams

with stream sizes to provide reasonable test durations. The VPP Host Stack QUIC transport is configured to utilize the picotls encryption library. In the future, tests utilizing addtional encryption algorithms will be added.

1.5.9 Reconfiguration Tests

Important: **DISCLAIMER**: Described reconf test methodology is experimental, and subject to change following consultation within csit-dev, vpp-dev and user communities. Current test results should be treated as indicative.

Overview

Reconf tests are designed to measure the impact of VPP re-configuration on data plane traffic. While VPP takes some measures against the traffic being entirely stopped for a prolonged time, the immediate forwarding rate varies during the re-configuration, as some configurations steps need the active dataplane worker threads to be stopped temporarily.

As the usual methods of measuring throughput need multiple trial measurements with somewhat long durations, and the re-configuration process can also be long, finding an offered load which would result in zero loss during the re-configuration process would be time-consuming.

Instead, reconf tests first find a througput value (lower bound for NDR) without re-configuration, and then maintain that ofered load during re-configuration. The measured loss count is then assumed to be caused by the re-configuration process. The result published by reconf tests is the effective blocked time, that is the loss count divided by the offered load.

Current Implementation

Each reconf suite is based on a similar MLRsearch performance suite.

MLRsearch parameters are changed to speed up the throughput discovery. For example, PDR is not searched for, and the final trial duration is shorter.

The MLRsearch suite has to contain a configuration parameter that can be scaled up, e.g. number of tunnels or number of service chains. Currently, only increasing the scale is supported as the re-configuration operation. In future, scale decrease or other operations can be implemented.

The traffic profile is not changed, so the traffic present is processed only by the smaller scale configuration. The added tunnels / chains are not targetted by the traffic.

For the re-configuration, the same Robot Framework and Python libraries are used, as were used in the initial configuration, with the exception of the final calls that do not interact with VPP (e.g. starting virtual machines) being skipped to reduce the test overall duration.

Discussion

Robot Framework introduces a certain overhead, which may affect timing of individual VPP API calls, which in turn may affect the number of packets lost.

The exact calls executed may contain unnecessary info dumps, repeated commands, or commands which change a value that do not need to be changed (e.g. MTU). Thus, implementation details are affecting the results, even if their effect on the corresponding MLRsearch suite is negligible.

The lower bound for NDR is the only value safe to be used when zero packets lost are expected without re-configuration. But different suites show different "jitter" in that value. For some suites, the lower bound is not tight, allowing full NIC buffers to drain quickly between worker pauses. For other suites, lower bound for NDR still has quite a large probability of non-zero packet loss even without re-configuration.

1.5.10 VPP Startup Settings

CSIT code manipulates a number of VPP settings in startup.conf for optimized performance. List of common settings applied to all tests and test dependent settings follows.

See VPP startup.conf[24] for a complete set and description of listed settings.

Common Settings

List of VPP startup.conf settings applied to all tests:

1. heap-size <value> - set separately for ip4, ip6, stats, main depending on scale tested.
2. no-tx-checksum-offload - disables UDP / TCP TX checksum offload in DPDK. Typically needed for use faster vector PMDs (together with no-multi-seg).
3. buffers-per-numa <value> - sets a number of memory buffers allocated to VPP per CPU socket. VPP default is 16384. Needs to be increased for scenarios with large number of interfaces and worker threads. To accommodate for scale tests, CSIT is setting it to the maximum possible value corresponding to the limit of DPDK memory mappings (currently 256). For Xeon Skylake platforms configured with 2MB hugepages and VPP data-size and buffer-size defaults (2048B and 2496B respectively), this results in value of 215040 (256 * 840 = 215040, 840 * 2496B buffers fit in 2MB hugepage). For Xeon Haswell nodes value of 107520 is used.

[24] https://git.fd.io/vpp/tree/src/vpp/conf/startup.conf?h=stable/1908_2&id=fce396738f865293f0a023bc7f172086f81da456

Per Test Settings

List of vpp startup.conf settings applied dynamically per test:

1. corelist-workers <list_of_cores> - list of logical cores to run VPP worker data plane threads. Depends on HyperThreading and core per test configuration.
2. num-rx-queues <value> - depends on a number of VPP threads and NIC interfaces.
3. no-multi-seg - disables multi-segment buffers in DPDK, improves packet throughput, but disables Jumbo MTU support. Disabled for all tests apart from the ones that require Jumbo 9000B frame support.
4. UIO driver - depends on topology file definition.
5. QAT VFs - depends on NRThreads, each thread = 1QAT VFs.

1.5.11 KVM VMs vhost-user

QEMU is used for KVM VM vhost-user testing enviroment. By default, standard QEMU version is used, preinstalled from OS repositories (qemu-2.11.1 for Ubuntu 18.04). The path to the QEMU binary can be adjusted in *Constants.py*.

FD.io CSIT performance lab is testing VPP vhost-user with KVM VMs using following environment settings:

CSIT supports two types of VMs:

- **Image-VM**: used for all functional, VPP_device, and regular performance tests except NFV density tests.
- **Kernel-VM**: new VM type introduced for NFV density tests to provide greater in-VM application install flexibility and to further reduce test execution time by simpler VM lifecycle management.

Image-VM

CSIT can use a pre-created VM image. The path to the image can be adjusted in *Constants.py*. For convenience and full compatibility CSIT repository contains a set of scripts to prepare Built-root[25] based embedded Linux image with all the dependencies needed to run DPDK Testpmd, DPDK L3Fwd, Linux bridge or Linux IPv4 forwarding.

Built-root was chosen for a VM image to make it lightweight and with fast booting time to limit impact on tests duration.

In order to execute CSIT tests, VM image must have following software installed: qemu-guest-agent, sshd, bridge-utils, VirtIO support and DPDK Testpmd/L3fwd applications. Username/password for the VM must be `cisco/cisco` and `NOPASSWD` sudo access. The interface naming is based on the driver (management interface type is Intel E1000), all E1000 interfaces will be named `mgmt<n>` and all VirtIO interfaces will be named `virtio<n>`. In VM `/etc/init.d/qemu-guest-agent` must be set to `TRANSPORT=isa-serial:/dev/ttyS1` because ttyS0 is used by serial console and ttyS1 is dedicated for qemu-guest-agent in QEMU setup.

Kernel-VM

CSIT can use a kernel KVM image as a boot kernel, as an alternative to image VM. This option allows better configurability of what application is running in VM userspace. Using root9p filesystem allows mapping the host-OS filesystem as read only guest-OS filesystem.

Example of custom init script for the kernel-VM:

[25] https://buildroot.org/

```
#!/bin/bash
mount -t sysfs -o "nodev,noexec,nosuid" sysfs /sys
mount -t proc -o "nodev,noexec,nosuid" proc /proc
mkdir /dev/pts
mkdir /dev/hugepages
mount -t devpts -o "rw,noexec,nosuid,gid=5,mode=0620" devpts /dev/pts || true
mount -t tmpfs -o "rw,noexec,nosuid,size=10%,mode=0755" tmpfs /run
mount -t tmpfs -o "rw,noexec,nosuid,size=10%,mode=0755" tmpfs /tmp
mount -t hugetlbfs -o "rw,relatime,pagesize=2M" hugetlbfs /dev/hugepages
echo 0000:00:06.0 > /sys/bus/pci/devices/0000:00:06.0/driver/unbind
echo 0000:00:07.0 > /sys/bus/pci/devices/0000:00:07.0/driver/unbind
echo vfio-pci > /sys/bus/pci/devices/0000:00:06.0/driver_override
echo vfio-pci > /sys/bus/pci/devices/0000:00:07.0/driver_override
echo 0000:00:06.0 > /sys/bus/pci/drivers/vfio-pci/bind
echo 0000:00:07.0 > /sys/bus/pci/drivers/vfio-pci/bind
$vnf_bin
poweroff -f
```

QemuUtils library during runtime replaces the `$vnf_bin` variable by the path to NF binary and its parameters. This allows CSIT to run any application installed on host OS, for example the same version of VPP as running on the host-OS.

Kernel-VM image must be available in the host filesystem as a prerequisite. The path to kernel-VM image is defined in *Constants.py*.

1.5.12 LXC/DRC Container Memif

CSIT includes tests taking advantage of VPP memif virtual interface (shared memory interface) to interconnect VPP running in Containers. VPP vswitch instance runs in bare-metal user-mode handling NIC interfaces and connecting over memif (Slave side) to VPPs running in Linux Container (LXC) or in Docker Container (DRC) configured with memif (Master side). LXCs and DRCs run in a priviliged mode with VPP data plane worker threads pinned to dedicated physical CPU cores per usual CSIT practice. All VPP instances run the same version of software. This test topology is equivalent to existing tests with vhost-user and VMs as described earlier in *Logical Topologies* (page 37).

In addition to above vswitch tests, a single memif interface test is executed. It runs in a simple topology of two VPP container instances connected over memif interface in order to verify standalone memif interface performance.

More information about CSIT LXC and DRC setup and control is available in *Container Orchestration in CSIT* (page 162).

1.5.13 NFV Service Density

Network Function Virtualization (NFV) service density tests focus on measuring total per server throughput at varied NFV service "packing" densities with vswitch providing host dataplane. The goal is to compare and contrast performance of a shared vswitch for different network topologies and virtualization technologies, and their impact on vswitch performance and efficiency in a range of NFV service configurations.

Each NFV service instance consists of a set of Network Functions (NFs), running in VMs (VNFs) or in Containers (CNFs), that are connected into a virtual network topology using VPP vswitch running in Linux user-mode. Multiple service instances share the vswitch that in turn provides per service chain forwarding context(s). In order to provide a most complete picture, each network topology and service configuration is tested in different service density setups by varying two parameters:

- Number of service instances (e.g. 1, 2, 4, 6, 8, 10).
- Number of NFs per service instance (e.g. 1, 2, 4, 6, 8, 10).

Implementation of NFV service density tests in CSIT-1908.2 is using two NF applications:

- VNF: VPP of the same version as vswitch running in KVM VM, configured with /8 IPv4 prefix routing.
- CNF: VPP of the same version as vswitch running in Docker Container, configured with /8 IPv4 prefix routing.

Tests are designed such that in all tested cases VPP vswitch is the most stressed application, as for each flow vswitch is processing each packet multiple times, whereas VNFs and CNFs process each packets only once. To that end, all VNFs and CNFs are allocated enough resources to not become a bottleneck.

Service Configurations

Following NFV network topologies and configurations are tested:

- VNF Service Chains (VSC) with L2 vswitch
  - *Network Topology*: Sets of VNFs dual-homed to VPP vswitch over virtio-vhost links. Each set belongs to separate service instance.
  - *Network Configuration*: VPP L2 bridge-domain contexts form logical service chains of VNF sets and connect each chain to physical interfaces.
- CNF Service Chains (CSC) with L2 vswitch
  - *Network Topology*: Sets of CNFs dual-homed to VPP vswitch over memif links. Each set belongs to separate service instance.
  - *Network Configuration*: VPP L2 bridge-domain contexts form logical service chains of CNF sets and connect each chain to physical interfaces.
- CNF Service Pipelines (CSP) with L2 vswitch
  - *Network Topology*: Sets of CNFs connected into pipelines over a series of memif links, with edge CNFs single-homed to VPP vswitch over memif links. Each set belongs to separate service instance.
  - *Network Configuration*: VPP L2 bridge-domain contexts connect each CNF pipeline to physical interfaces.

Thread-to-Core Mapping

CSIT defines specific ratios for mapping software threads of vswitch and VNFs/CNFs to physical cores, with separate ratios defined for main control threads and data-plane threads.

In CSIT-1908.2 NFV service density tests run on Intel Xeon testbeds with Intel Hyper-Threading enabled, so each physical core is associated with a pair of sibling logical cores corresponding to the hyper-threads.

CSIT-1908.2 executes tests with the following software thread to physical core mapping ratios:

- vSwitch
  - Data-plane on single core
    - (main:core) = (1:1) => 1mt1c - 1 main thread on 1 core.
    - (data:core) = (1:1) => 2dt1c - 2 Data-plane Threads on 1 Core.
  - Data-plane on two cores
    - (main:core) = (1:1) => 1mt1c - 1 Main Thread on 1 Core.
    - (data:core) = (1:2) => 4dt2c - 4 Data-plane Threads on 2 Cores.
- VNF and CNF
  - Data-plane on single core

  * (main:core) = (2:1) => 2mt1c - 2 Main Threads on 1 Core, 1 Thread per NF, core shared between two NFs.
  * (data:core) = (1:1) => 2dt1c - 2 Data-plane Threads on 1 Core per NF.
- Data-plane on single logical core (Two NFs per physical core)
  * (main:core) = (2:1) => 2mt1c - 2 Main Threads on 1 Core, 1 Thread per NF, core shared between two NFs.
  * (data:core) = (2:1) => 2dt1c - 2 Data-plane Threads on 1 Core, 1 Thread per NF, core shared between two NFs.

Maximum tested service densities are limited by a number of physical cores per NUMA. CSIT-1908.2 allocates cores within NUMA0. Support for multi NUMA tests is to be added in future release.

1.5.14 VPP_Device Functional

CSIT-1908.2 includes VPP_Device test environment for functional VPP device tests integrated into LFN CI/CD infrastructure. VPP_Device tests run on 1-Node testbeds (1n-skx, 1n-arm) and rely on Linux SRIOV Virtual Function (VF), dot1q VLAN tagging and external loopback cables to facilitate packet passing over external physical links. Initial focus is on few baseline tests. New device tests can be added by small edits to existing CSIT Performance (2-node) test. RF test definition code stays unchanged with the exception of traffic generator related L2 KWs.

1.5.15 IPSec on Intel QAT

VPP IPSec performance tests are using DPDK cryptodev device driver in combination with HW cryptodev devices - Intel QAT 8950 50G - present in LF FD.io physical testbeds. DPDK cryptodev can be used for all IPSec data plane functions supported by VPP.

Currently CSIT-1908.2 implements following IPSec test cases:

- AES-GCM, CBC-SHA1 ciphers, in combination with IPv4 routed-forwarding with Intel xl710 NIC.
- CBC-SHA1 ciphers, in combination with LISP-GPE overlay tunneling for IPv4-over-IPv4 with Intel xl710 NIC.

1.5.16 TRex Traffic Generator

Usage

TRex traffic generator[26] is used for all CSIT performance tests. TRex stateless mode is used to measure NDR and PDR throughputs using MLRsearch and to measure maximum transer rate in MRR tests.

TRex is installed and run on the TG compute node. The typical procedure is:

- If the TRex is not already installed on TG, it is installed in the suite setup phase - see TRex installation[27].
- TRex configuration is set in its configuration file

```
/etc/trex_cfg.yaml
```

- TRex is started in the background mode

```
$ sh -c 'cd <t-rex-install-dir>/scripts/ && sudo nohup ./t-rex-64 -i --prefix $(hostname) --
↪hdrh --no-scapy-server > /tmp/trex.log 2>&1 &' > /dev/null
```

[26] https://trex-tgn.cisco.com
[27] https://git.fd.io/csit/tree/resources/tools/trex/trex_installer.sh?h=rls1908_2

- There are traffic streams dynamically prepared for each test, based on traffic profiles. The traffic is sent and the statistics obtained using `trex.stl.api.STLClient`.

Measuring Packet Loss

Following sequence is followed to measure packet loss:

- Create an instance of STLClient.
- Connect to the client.
- Add all streams.
- Clear statistics.
- Send the traffic for defined time.
- Get the statistics.

If there is a warm-up phase required, the traffic is sent also before test and the statistics are ignored.

Measuring Latency

If measurement of latency is requested, two more packet streams are created (one for each direction) with TRex flow_stats parameter set to STLFlowLatencyStats. In that case, returned statistics will also include min/avg/max latency values and encoded HDRHstogram data.

CHAPTER

TWO

VPP PERFORMANCE

2.1 Overview

VPP performance test results are reported for all three physical testbed types present in FD.io labs: 3-Node Xeon Haswell (3n-hsw), 3-Node Xeon Skylake (3n-skx), 2-Node Xeon Skylake (2n-skx) and installed NIC models. For description of physical testbeds used for VPP performance tests please refer to *Physical Testbeds* (page 3).

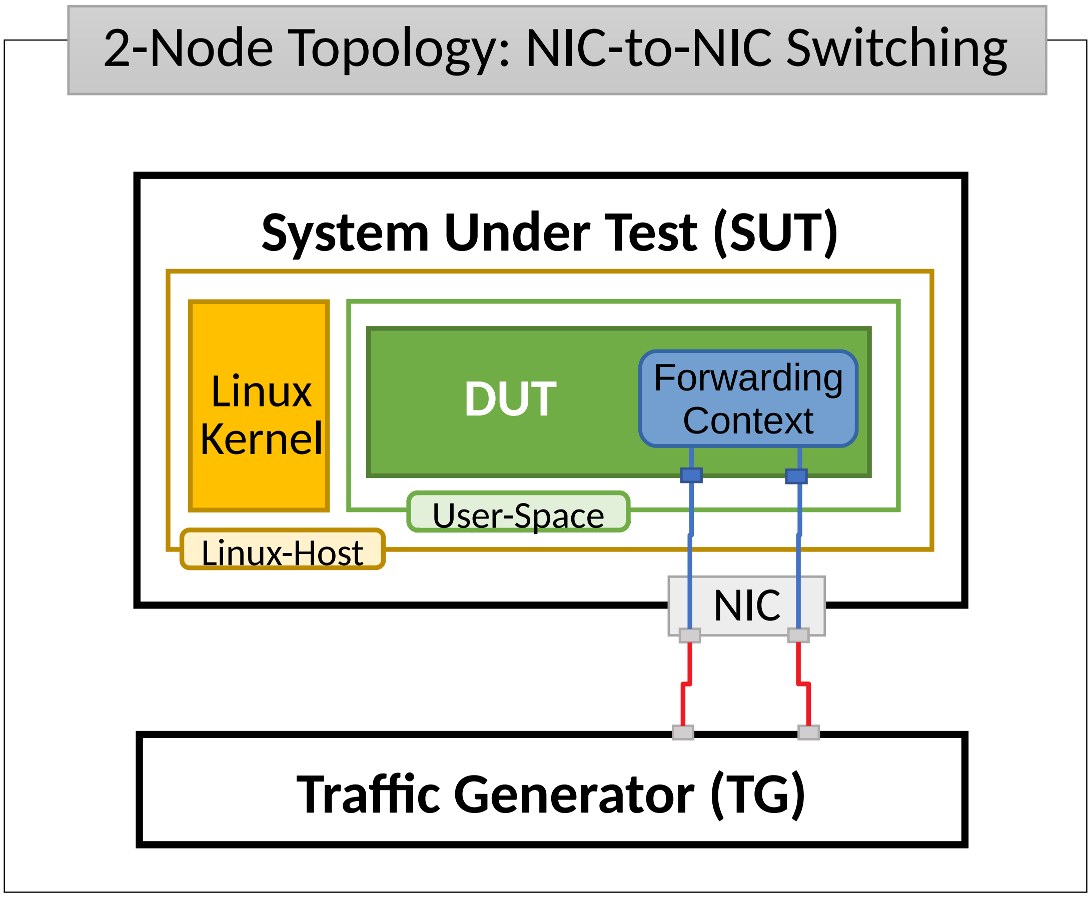
2.1.1 Logical Topologies

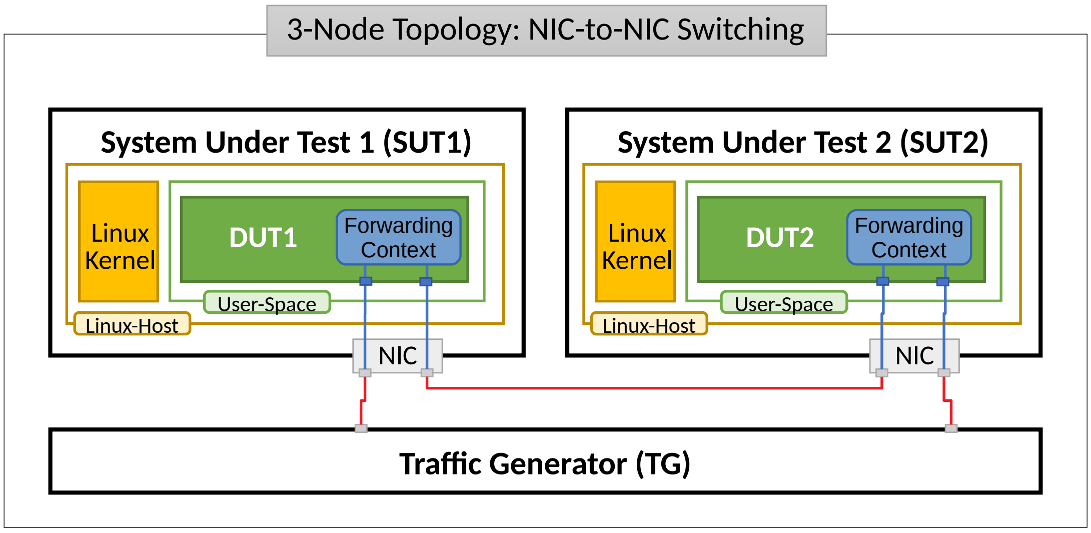
CSIT VPP performance tests are executed on physical testbeds described in *Physical Testbeds* (page 3). Based on the packet path thru server SUTs, three distinct logical topology types are used for VPP DUT data plane testing:

1. NIC-to-NIC switching topologies.
2. VM service switching topologies.
3. Container service switching topologies.

NIC-to-NIC Switching

The simplest logical topology for software data plane application like VPP is NIC-to-NIC switching. Tested topologies for 2-Node and 3-Node testbeds are shown in figures below.





Server Systems Under Test (SUT) run VPP application in Linux user-mode as a Device Under Test (DUT). Server Traffic Generator (TG) runs T-Rex application. Physical connectivity between SUTs and TG is provided using different drivers and NIC models that need to be tested for performance (packet/bandwidth throughput and latency).

From SUT and DUT perspectives, all performance tests involve forwarding packets between two (or more) physical Ethernet ports (10GE, 25GE, 40GE, 100GE). In most cases both physical ports on SUT are located on the same NIC. The only exceptions are link bonding and 100GE tests. In the latter case only one port per NIC can be driven at linerate due to PCIe Gen3 x16 slot bandwidth limiations. 100GE NICs are not supported in PCIe Gen3 x8 slots.

Note that reported VPP DUT performance results are specific to the SUTs tested. SUTs with other processors than the ones used in FD.io lab are likely to yield different results. A good rule of thumb, that can be applied to estimate VPP packet thoughput for NIC-to-NIC switching topology, is to expect the forwarding performance to be proportional to processor core frequency for the same processor architecture, assuming processor is the only limiting factor and all other SUT parameters are equivalent to FD.io CSIT environment.

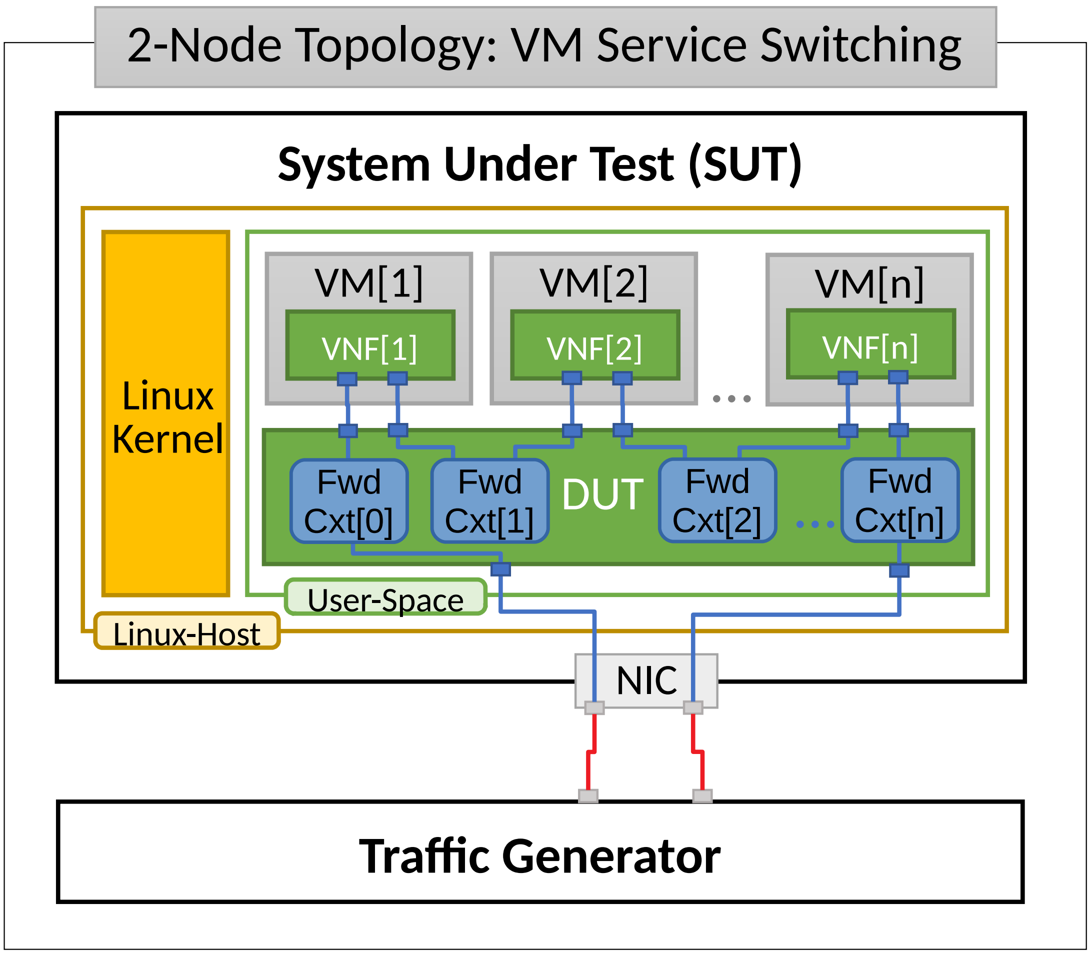
VM Service Switching

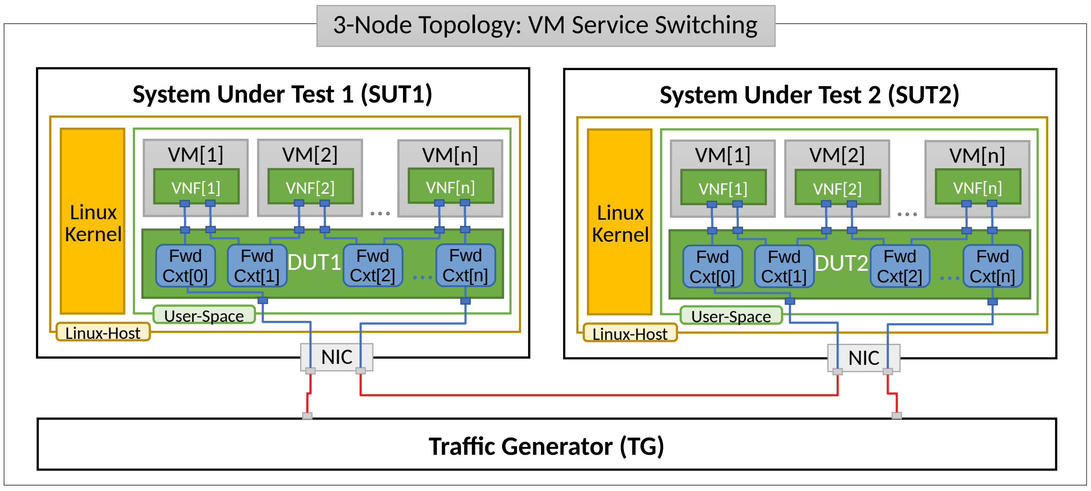
VM service switching topology test cases require VPP DUT to communicate with Virtual Machines (VMs) over vhost-user virtual interfaces.

Two types of VM service topologies are tested in CSIT-1908.2:

1. "Parallel" topology with packets flowing within SUT from NIC(s) via VPP DUT to VM, back to VPP DUT, then out thru NIC(s).
2. "Chained" topology (a.k.a. "Snake") with packets flowing within SUT from NIC(s) via VPP DUT to VM, back to VPP DUT, then to the next VM, back to VPP DUT and so on and so forth until the last VM in a chain, then back to VPP DUT and out thru NIC(s).

For each of the above topologies, VPP DUT is tested in a range of L2 or IPv4/IPv6 configurations depending on the test suite. Sample VPP DUT "Chained" VM service topologies for 2-Node and 3-Node testbeds with each SUT running N of VM instances is shown in the figures below.





In "Chained" VM topologies, packets are switched by VPP DUT multiple times: twice for a single VM, three times for two VMs, N+1 times for N VMs. Hence the external throughput rates measured by TG and listed in this report must be multiplied by N+1 to represent the actual VPP DUT aggregate packet forwarding rate.

For "Parallel" service topology packets are always switched twice by VPP DUT per service chain.

Note that reported VPP DUT performance results are specific to the SUTs tested. SUTs with other processor than the ones used in FD.io lab are likely to yield different results. Similarly to NIC-to-NIC switching topology, here one can also expect the forwarding performance to be proportional to processor core frequency for the same processor architecture, assuming processor is the only limiting factor. However due to much higher dependency on intensive memory operations in VM service chained topologies and sensitivity to Linux scheduler settings and behaviour, this estimation may not always yield good enough accuracy.

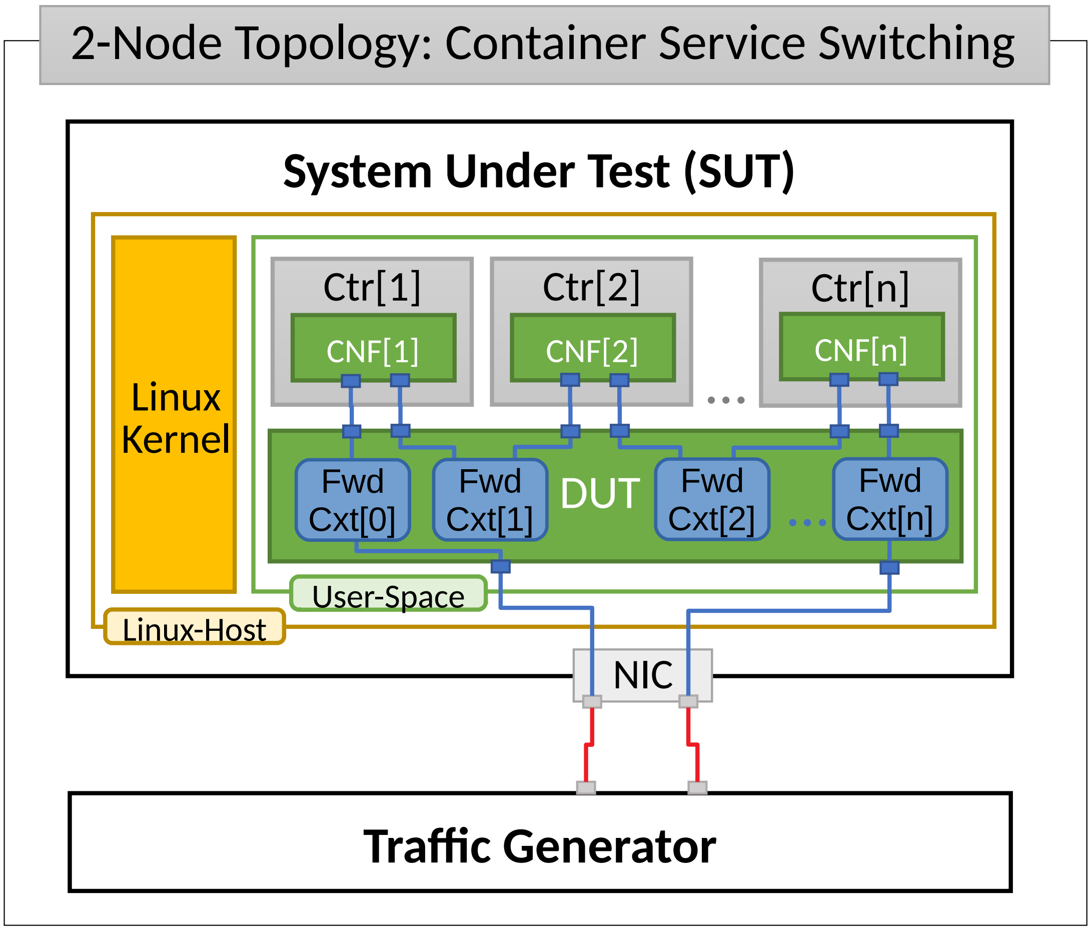
Container Service Switching

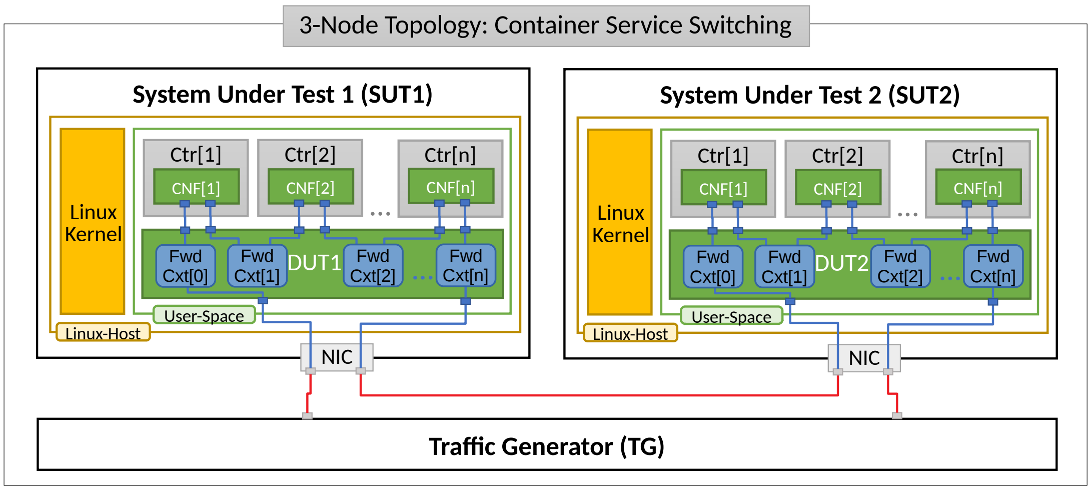
Container service switching topology test cases require VPP DUT to communicate with Containers (Ctrs) over memif virtual interfaces.

Three types of VM service topologies are tested in CSIT-1908.2:

1. "Parallel" topology with packets flowing within SUT from NIC(s) via VPP DUT to Container, back to VPP DUT, then out thru NIC(s).
2. "Chained" topology (a.k.a. "Snake") with packets flowing within SUT from NIC(s) via VPP DUT to Container, back to VPP DUT, then to the next Container, back to VPP DUT and so on and so forth until the last Container in a chain, then back to VPP DUT and out thru NIC(s).
3. "Horizontal" topology with packets flowing within SUT from NIC(s) via VPP DUT to Container, then via "horizontal" memif to the next Container, and so on and so forth until the last Container, then back to VPP DUT and out thru NIC(s).

For each of the above topologies, VPP DUT is tested in a range of L2 or IPv4/IPv6 configurations depending on the test suite. Sample VPP DUT "Chained" Container service topologies for 2-Node and 3-Node testbeds with each SUT running N of Container instances is shown in the figures below.





In "Chained" Container topologies, packets are switched by VPP DUT multiple times: twice for a single Container, three times for two Containers, N+1 times for N Containers. Hence the external throughput rates measured by TG and listed in this report must be multiplied by N+1 to represent the actual VPP DUT aggregate packet forwarding rate.

For a "Parallel" and "Horizontal" service topologies packets are always switched by VPP DUT twice per service chain.

Note that reported VPP DUT performance results are specific to the SUTs tested. SUTs with other processor than the ones used in FD.io lab are likely to yield different results. Similarly to NIC-to-NIC switching topology, here one can also expect the forwarding performance to be proportional to processor core frequency for the same processor architecture, assuming processor is the only limiting factor. However due to much higher dependency on intensive memory operations in Container service chained topologies and sensitivity to Linux scheduler settings and behaviour, this estimation may not always yield good enough accuracy.

2.1.2 Performance Tests Coverage

Performance tests measure following metrics for tested VPP DUT topologies and configurations:

- Packet Throughput: measured in accordance with **RFC 2544**[28], using FD.io CSIT Multiple Loss Ratio search (MLRsearch), an optimized binary search algorithm, producing throughput at different Packet Loss Ratio (PLR) values:
  - Non Drop Rate (NDR): packet throughput at PLR=0%.
  - Partial Drop Rate (PDR): packet throughput at PLR=0.5%.
- One-Way Packet Latency: measured at different offered packet loads:
  - 100% of discovered NDR throughput.
  - 100% of discovered PDR throughput.
- Maximum Receive Rate (MRR): measure packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss. Maximum load for specified Ethernet frame size is set to the bi-directional link rate.

CSIT-1908.2 includes following VPP data plane functionality performance tested across a range of NIC drivers and NIC models:

[28] https://tools.ietf.org/html/rfc2544.html

Functionality	Description
ACL	L2 Bridge-Domain switching and IPv4and IPv6 routing with iACL and oACL IP address, MAC address and L4 port security.
COP	IPv4 and IPv6 routing with COP address security.
IPv4	IPv4 routing.
IPv6	IPv6 routing.
IPv4 Scale	IPv4 routing with 20k, 200k and 2M FIB entries.
IPv6 Scale	IPv6 routing with 20k, 200k and 2M FIB entries.
IPSecHW	IPSec encryption with AES-GCM, CBC-SHA-256 ciphers, in combination with IPv4 routing. Intel QAT HW acceleration.
IPSec+LISP	IPSec encryption with CBC-SHA1 ciphers, in combination with LISP-GPE overlay tunneling for IPv4-over-IPv4.
IPSecSW	IPSec encryption with AES-GCM, CBC-SHA-256 ciphers, in combination with IPv4 routing.
KVM VMs vhost-user	Virtual topologies with service chains of 1 VM using vhost-user interfaces, with different VPP forwarding modes incl. L2XC, L2BD, VXLAN with L2BD, IPv4 routing.
L2BD	L2 Bridge-Domain switching of untagged Ethernet frames with MAC learning; disabled MAC learning i.e. static MAC tests to be added.
L2BD Scale	L2 Bridge-Domain switching of untagged Ethernet frames with MAC learning; disabled MAC learning i.e. static MAC tests to be added with 20k, 200k and 2M FIB entries.
L2XC	L2 Cross-Connect switching of untagged, dot1q, dot1ad VLAN tagged Ethernet frames.
LISP	LISP overlay tunneling for IPv4-over-IPv4, IPv6-over-IPv4, IPv6-over-IPv6, IPv4-over-IPv6 in IPv4 and IPv6 routing modes.
LXC/DRC Containers Memif	Container VPP memif virtual interface tests with different VPP forwarding modes incl. L2XC, L2BD.
NAT	(Source) Network Address Translation tests with varying number of users and ports per user.
QoS Policer	Ingress packet rate measuring, marking and limiting (IPv4).
SRv6 Routing	Segment Routing IPv6 tests.
VPP TCP/IP stack	Tests of VPP TCP/IP stack used with VPP built-in HTTP server.
VTS	Virtual Topology System use case tests combining VXLAN overlay tunneling with L2BD, ACL and KVM VM vhost-user features.
VXLAN	VXLAN overlay tunnelling integration with L2XC and L2BD.

Execution of performance tests takes time, especially the throughput tests. Due to limited HW testbed resources available within FD.io labs hosted by LF, the number of tests for some NIC models has been limited to few baseline tests.

2.1.3 Performance Tests Naming

FD.io CSIT-1908.2 follows a common structured naming convention for all performance and system functional tests, introduced in CSIT-17.01.

The naming should be intuitive for majority of the tests. Complete description of FD.io CSIT test naming convention is provided on *Test Naming* (page 185).

2.2 Release Notes

2.2.1 Changes in CSIT-1908.2

1. VPP PERFORMANCE TESTS
   - **Intel Xeon 2n-skx, 3n-skx testbeds**: VPP performance test data is provided using a different CSIT test environment compared to CSIT-1908.1 and CSIT-1908. The changes were applied during the CSIT-2001 development cycle.
     - CSIT test environment is now versioned, with ver. 1 associated with CSIT rls1908 git branch as of 2019-08-21, and ver. 2 associated with CSIT rls2001 git branch as of 2020-03-27.
     - To identify performance changes due to VPP code changes from v19.08.1 to v19.08.2, both have been tested in CSIT environment ver. 2. See *Current vs. Previous Release* (page 121) and *Known Issues* (page 45).
   - **Intel Xeon 2n-clx testbeds**: VPP performance test data is now included in this report. See *Known Issues* (page 45).
   - **Service density 2n-skx tests**: Added higher NF density tests with 802.1q (vlan) and VXLAN encapsulation from Traffic Generator.
   - **GBP tests**: Added GBP (Group Based Policy) routing test cases with 802.1q (vlan) external traffic.
   - **AVF IPv4 scale tests**: Increased coverage of AVF IPv4 base and scale test cases (Fortville NICs only).
   - **2n-skx tests**: Increased coverage of selected (COP, iACL, Policer) test cases.
   - **IPsec scale tests**: Added IPsec interface mode scale tests with 1, 40, 400, 1000, 5000, 10000, 20000, 40000, 60000 IPsec tunnels. Removed DPDK backend dependency. Major IPsec test code refactoring.
   - **Hoststack TCP/IP tests**: Major refactor of Hoststack TCP performance tests using WRK generator talking to the VPP HTTP static server plugin measuring connections per second and requests per second.
   - **Changed methodology of dot1q tests in 2-Node testbeds**: dot1q encapsulation is now used on both links of SUT. Previously dot1q was used only on a single link with the other link carrying untagged Ethernet frames. This change results in slightly lower throughput in CSIT-1908 for all dot1q tests in all 2-Node testbeds.
   - **KVM VM vhost-user tests**: completed move to Kernel-VM for all tests. In addition to running DPDK Testpmd as VM workload, new tests created with VPP as VM workload. VPP in VM is the same version as the DUT VPP (acting as vSwitch) and its configuration depends on the test type. For all L2 Ethernet Switching tests it's vpp-l2xc (L2 cross-connect), for all IPv4 Routing tests it's vpp-ip4 (VPP IPv4 routing).
2. TEST FRAMEWORK
   - **CSIT PAPI Support**: Finished conversion of CSIT VAT L1 keywords to PAPI L1 KWs in CSIT using VPP Python bindings (VPP PAPI). Redesign of key components of PAPI Socket Executor and PAPI history. Due to issues with PAPI performance, VAT is still used in CSIT for all VPP scale tests. See known issues below.
   - **General Code Housekeeping**: Ongoing RF keywords optimizations, removal of redundant RF keywords and aligning of suite/test setup/teardowns.
3. PRESENTATION AND ANALYTICS LAYER
   - **Graphs Layout Improvements**: Improved performance graphs layout for better readibility and maintenance: test grouping, axis labels, descriptions, other informative decoration.

2.2.2 Known Issues

List of known issues in CSIT-1908.2 for VPP performance tests:

#	JiraID	Issue Description
1	CSIT-570[29]	Sporadic (1 in 200) NDR discovery test failures on x520. DPDK reporting rx-errors, indicating L1 issue. Suspected issue with HW combination of X710-X520 in LF testbeds. Not observed outside of LF testbeds.
2	VPP-662[30]	9000B packets not supported by NICs VIC1227 and VIC1387.
3	CSIT-1498[31]	Memif tests are sporadically failing on initialization of memif connection.
4	VPP-1676[32]	9000B ip4 memif errors - ip4-input: ip4 length > l2 length. IP4 jumbo frames (9000B) are dropped in case of tests with memif.
5	VPP-1677[33]	9000B ip4 nat44: VPP crash + coredump. VPP crashes very often in case that NAT44 is configured and it has to process IP4 jumbo frames (9000B).
6	CSIT-1591[34] VPP-1763[35]	All CSIT scale tests can not use PAPI due to much slower performance compared to VAT/CLI (it takes much longer to program VPP). This needs to be addressed on the PAPI side.
7	CSIT-1593[36]	IPv4 AVF 9000B packet tests are failing on 3n-skx while passing on 2n-skx.

[29] https://jira.fd.io/browse/CSIT-570
[30] https://jira.fd.io/browse/VPP-662
[31] https://jira.fd.io/browse/CSIT-1498
[32] https://jira.fd.io/browse/VPP-1676
[33] https://jira.fd.io/browse/VPP-1677
[34] https://jira.fd.io/browse/CSIT-1591
[35] https://jira.fd.io/browse/VPP-1763
[36] https://jira.fd.io/browse/CSIT-1593

2.3 Packet Throughput

Throughput graphs are generated based on the results data obtained from the CSIT-1908.2 test jobs. In order to verify benchmark results repeatibility selected, CSIT performance tests are executed multiple times (target: 10 times) on each physical testbed type. Box-and-Whisker plots are used to display variations in measured throughput values.

Lists of tests selected for multiple execution and graphing are captured per testbed type in test_select_list_{testbed_type}.md[37] files.

Graphs are split into sections as follows:

1. **Header 1**: VPP packet path and lookup types
   - **L2 Ethernet Switching**: L2 bridge-doman, L2 cross-connect and L2 patch
   - **IPv4 Routing**: IPv4 routing with /32 prefixes
   - **IPv6 Routing**: IPv6 routing with /128 prefixes
   - **SRv6 Routing**: SRv6 with IPv6 routing
   - **IPv4 Tunnels**: IPv4 overlay tunnels
   - **KVM VMs vhost-user**: KVM VMs connected over virtio and vhost-user interfaces
   - **LXC/DRC Container Memif**: Linux containers and Docker containers connected over Memif interfaces
   - **IPsec IPv4 Routing**: IPsec encryption/decryption with IPv4 routing
   - **Virtual Topology System**: VXLAN configurations with L2 bridge-domains
2. **Header 2**: testbeds and NIC models
   - section name format:
     - {**testbed_type**}-{**nic_model**}
   - **testbed_type**:
     - 2n-skx: 2-node Xeon Skylake
     - 3n-skx: 3-node Xeon Skylake
     - 2n-clx: 2-node Xeon Cascade Lake
     - 3n-hsw: 3-node Xeon Haswell
     - 3n-tsh: 3-node Arm TaiShan
     - 2n-dnv: 2-node Atom Denverton
     - 3n-dnv: 3-node Atom Denverton
   - **nic_model**:
     - xxv710: xxv710 2p25GE Intel (Fortville)
     - x710: x710 4p10GE Intel (Fortville)
     - xl710: xl710 2p40GE Intel (Fortville)
     - x520: x520 2p10GE Intel (Niantic)
     - x553: x553 2p10GE Intel (Niantic)
3. **Header 3**: test group names
   - section name format:

[37] https://git.fd.io/csit/tree/docs/job_specs

  - {**frame_size**}-{**worker_thread_core_cfg**}-{**vpp_functionality**}-{**vpp_lookup_type**}-{**baseline_scale**}-{**nic_driver**}

- **frame_size**:
  - 64b: 64 byte frames, smallest frame size for untagged IPv4 packets
  - 78b: 78 byte frames, smallest frame size for untagged IPv6 packets
  - 114b: VXLAN encapsulated L2 frames
  - imix: a sequence of (7x64B, 4x570, 1x1518) byte frames
- **worker_thread_core_cfg**:
  - 1t1c: 1 worker thread on 1 core, hyper-threading not used
  - 2t1c: 2 worker threads on 1 core, hyper-threading used
- **vpp_functionality** (optional):
  - features: including input-acl, output-acl, macip-iacl, nat44
  - srv6: srv6 encap/decap, proxy
  - link-bonding: L2 link aggregation with 1 or 2 bonded links
  - ipsec: IPsec encryption/decryption with different ciphers
  - vts: Virtual Topology System specific tests
- **vpp_lookup_type**:
  - l2switching, ip4routing, ip6routing, ip4tunnel, vhost, memif
- **baseline_scale**:
  - base: baseline tests with less than 10 forwarding entries
  - scale: scale tests with up to 2 million forwarding entries
  - base-scale: both baseline and scale tests grouped together
- **nic_driver**:
  - avf: VPP native avf driver for Intel Fortville NICs
  - i40e: dpdk poll mode driver for Intel Fortville NICs
  - ixgbe: dpdk poll mode driver for Intel Niantic NICs

For each test case, Box-and-Whisker plots show the quartiles (Min, 1st quartile / 25th percentile, 2nd quartile / 50th percentile / mean, 3rd quartile / 75th percentile, Max) across collected data set. Outliers are plotted as individual points.

Additional information about graph data:

1. **Graph Title**: describes tested packet path, testbed topology, processor model, NIC model, packet size, number of cores and threads used by data plane workers and indication of VPP DUT configuration.
2. **X-axis Labels**: indices of individual test suites as listed in Graph Legend.
3. **Y-axis Labels**: measured Packets Per Second [pps] throughput values.
4. **Graph Legend**: lists X-axis indices with associated CSIT test suites executed to generate graphed test results.
5. **Hover Information**: lists minimum, first quartile, median, third quartile, and maximum. If either type of outlier is present the whisker on the appropriate side is taken to 1.5×IQR from the quartile (the “inner fence”) rather than the max or min, and individual outlying data points are displayed as unfilled circles (for suspected outliers) or filled circles (for outliers). (The “outer fence” is 3×IQR from the quartile.)

Note: Test results have been generated by FD.io test executor vpp performance job 2n-skx[38], FD.io test executor vpp performance job 3n-skx[39], FD.io test executor vpp performance job 2n-clx[40] with RF result files csit-vpp-perf-1908_2-*.zip archived here. Required per test case data set size is **10**, but for VPP tests the actual size varies per test case and is <=10.

[38] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-skx
[39] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-3n-skx
[40] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-clx

2.3.1 L2 Ethernet Switching

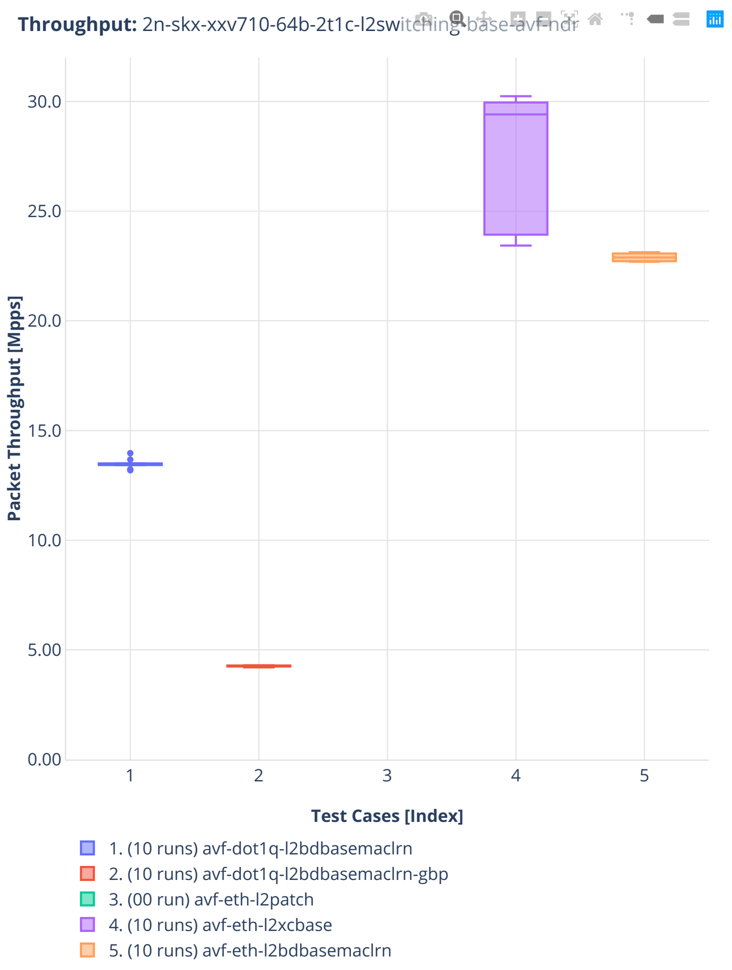
Following sections include summary graphs of VPP Phy-to-Phy performance with L2 Ethernet switching, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

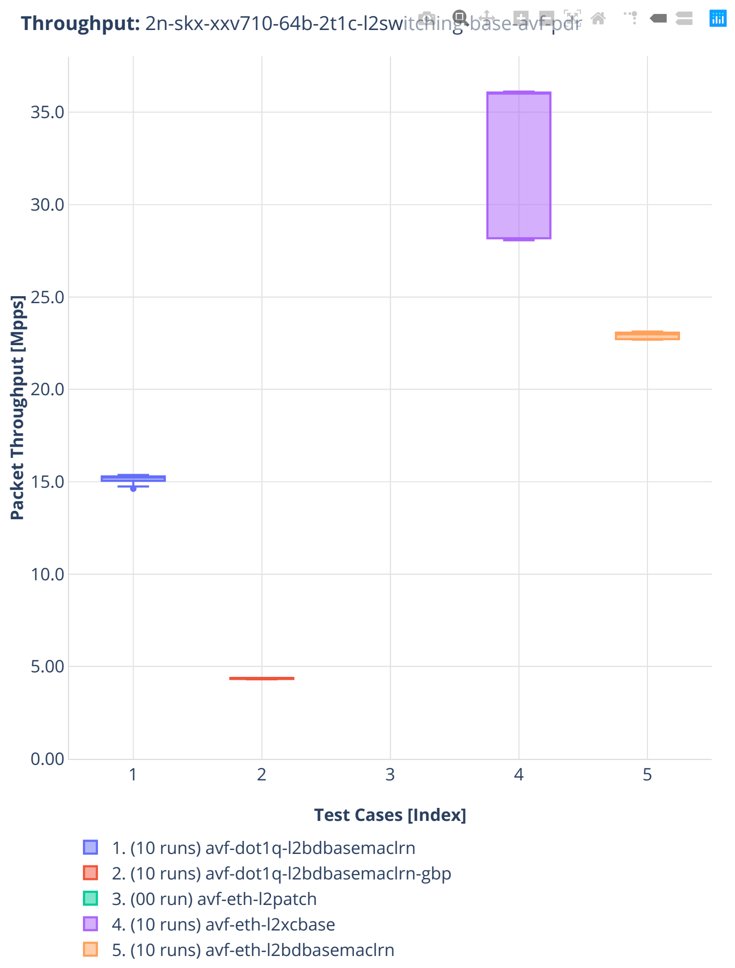
CSIT source code for the test cases used for plots can be found in CSIT git repository[41].

[41] https://git.fd.io/csit/tree/tests/vpp/perf/l2?h=rls1908_2

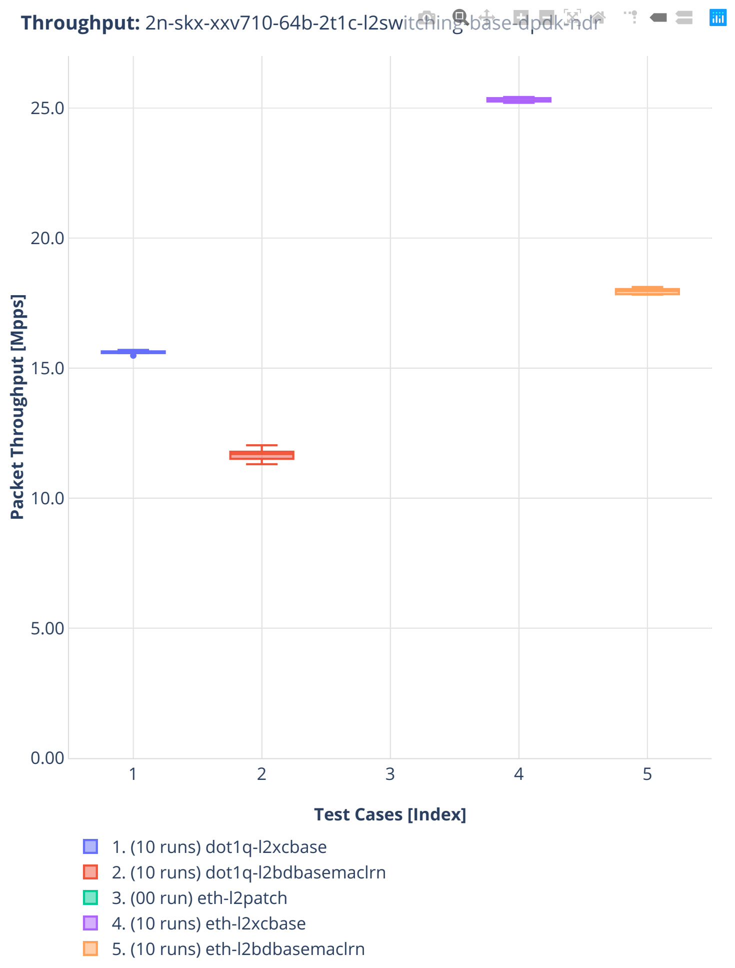
2n-skx-xxv710

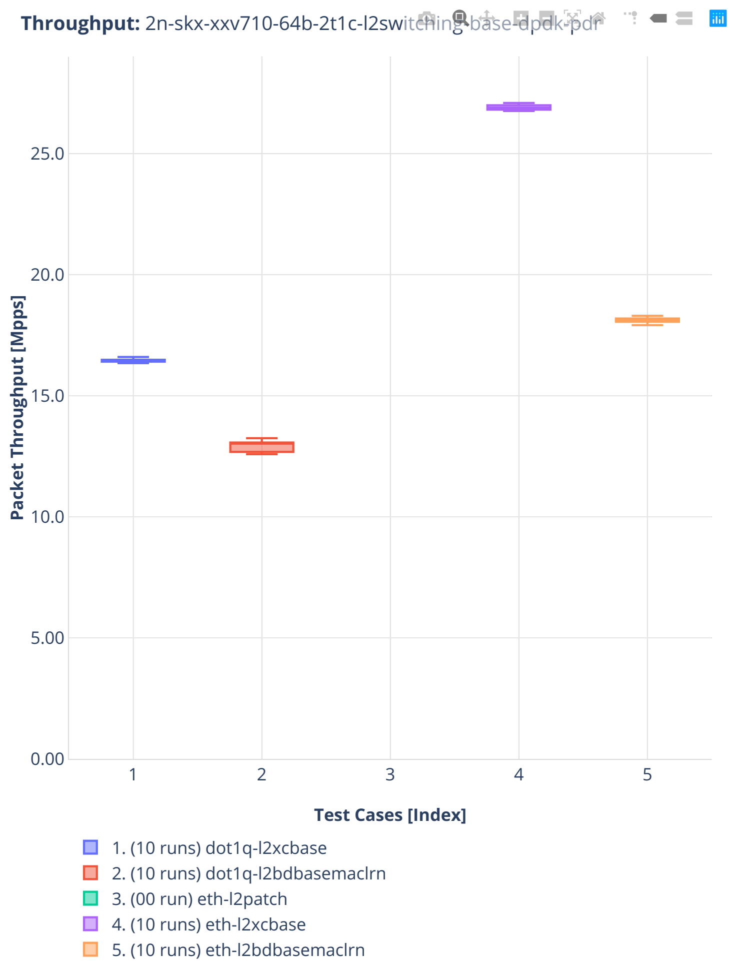
64b-2t1c-l2switching-base-avf



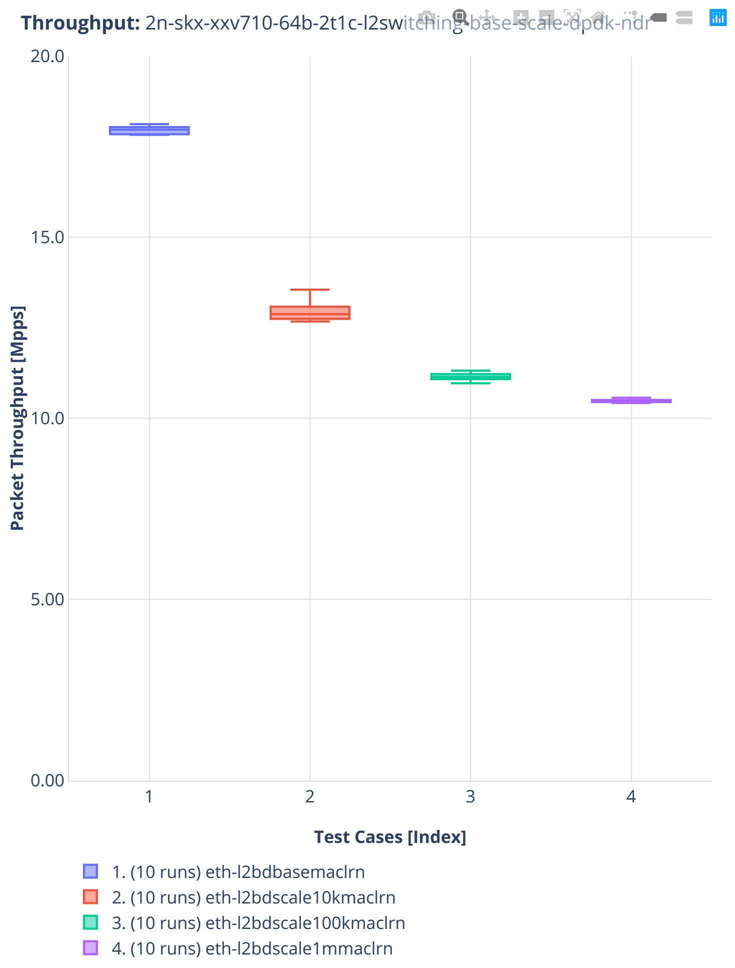

Throughput: 2n-skx-xxv710-64b-2t1c-l2switching-base-avf-pdr
Packet Throughput [Mpps]
35.0
30.0
25.0
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (10 runs) avf-dot1q-l2bdbasemaclrn
2. (10 runs) avf-dot1q-l2bdbasemaclrn-gbp
3. (00 run) avf-eth-l2patch
4. (10 runs) avf-eth-l2xcbase
5. (10 runs) avf-eth-l2bdbasemaclrn

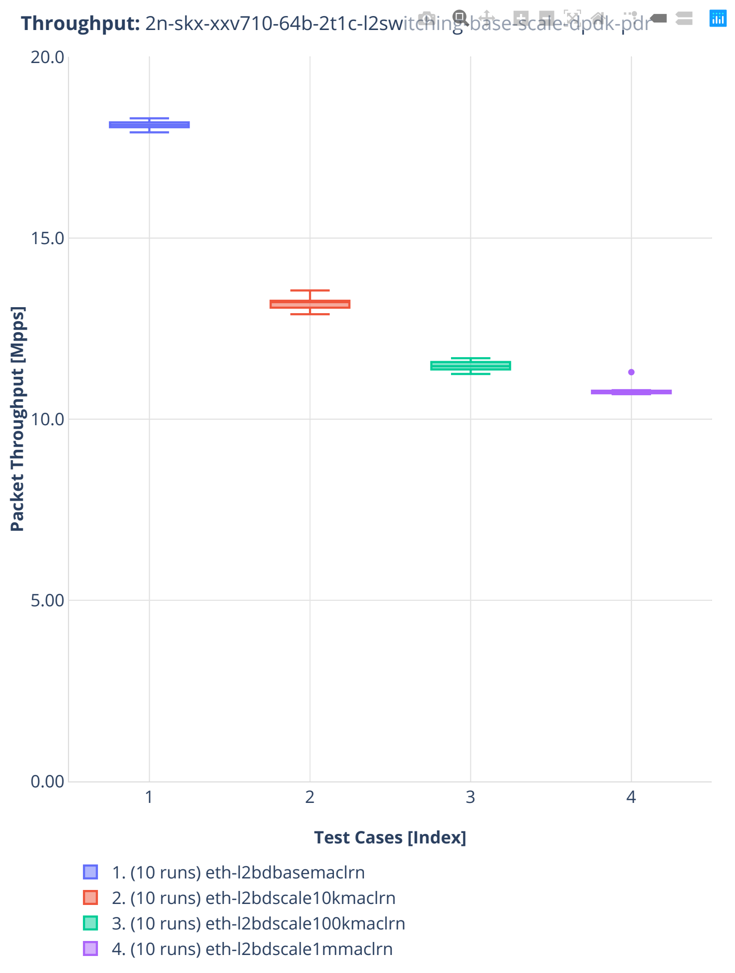
64b-2t1c-l2switching-base-dpdk




Throughput: 2n-skx-xxv710-64b-2t1c-l2switching-base-dpdk-pdr
Packet Throughput [Mpps]
25.0
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (10 runs) dot1q-l2xcbase
2. (10 runs) dot1q-l2bdbasemaclrn
3. (00 run) eth-l2patch
4. (10 runs) eth-l2xcbase
5. (10 runs) eth-l2bdbasemaclrn

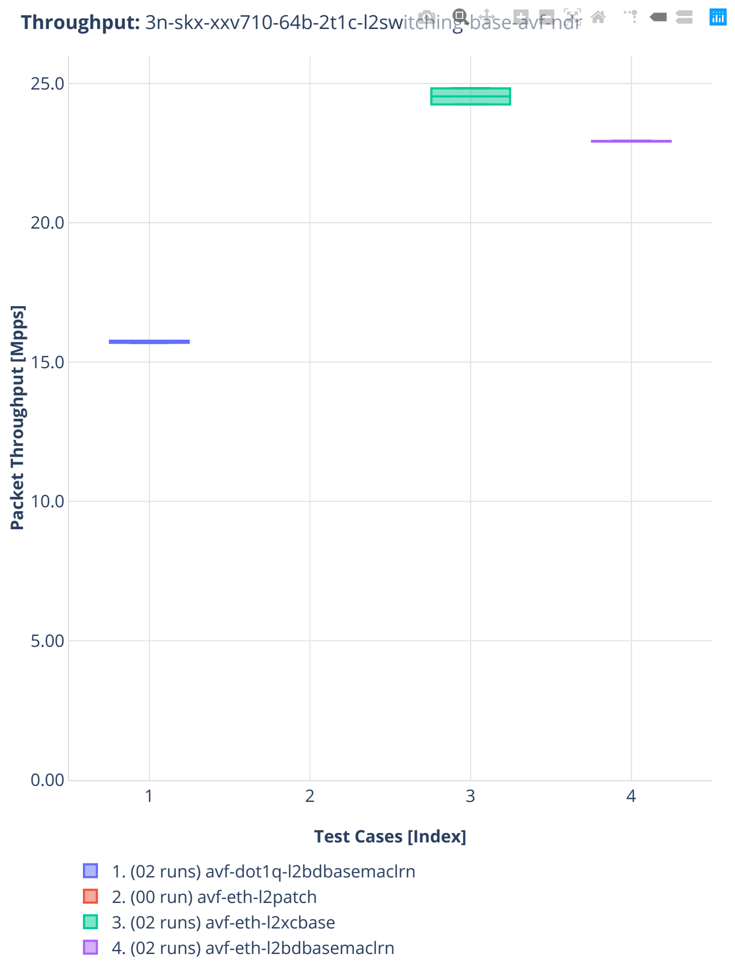
64b-2t1c-l2switching-base-scale-dpdk

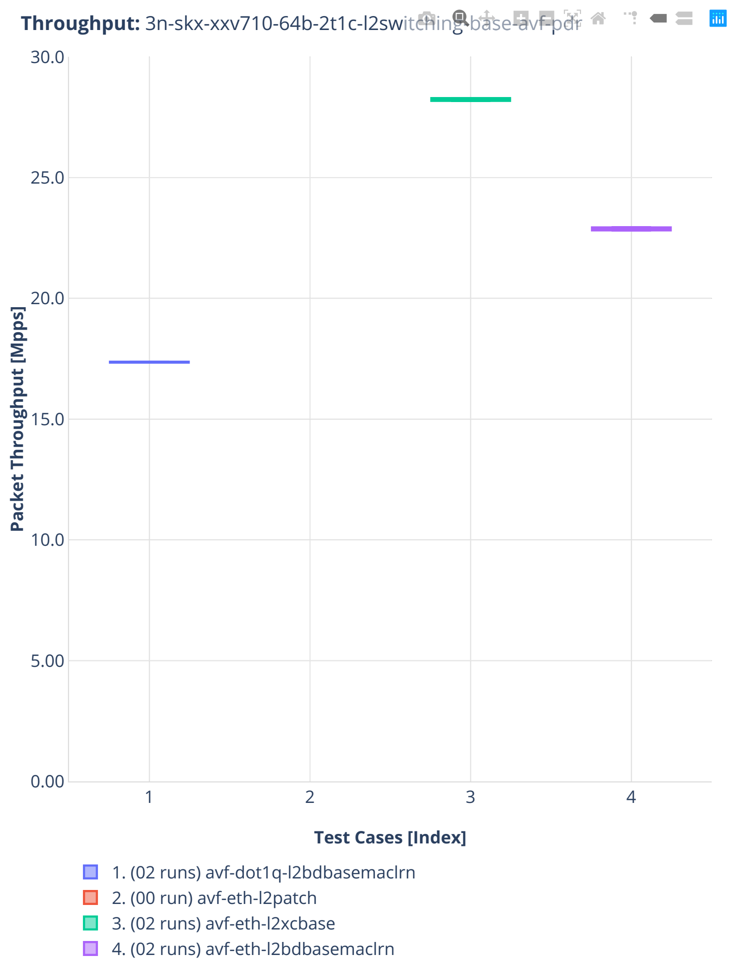



Throughput: 2n-skx-xxv710-64b-2t1c-l2switching-base-scale-dpdk-pdr
Packet Throughput [Mpps]
20.0
15.0
10.0
5.00
0.00
1
2
3
4
Test Cases [Index]
1. (10 runs) eth-l2bdbasemaclrn
2. (10 runs) eth-l2bdscale10kmaclrn
3. (10 runs) eth-l2bdscale100kmaclrn
4. (10 runs) eth-l2bdscale1mmaclrn

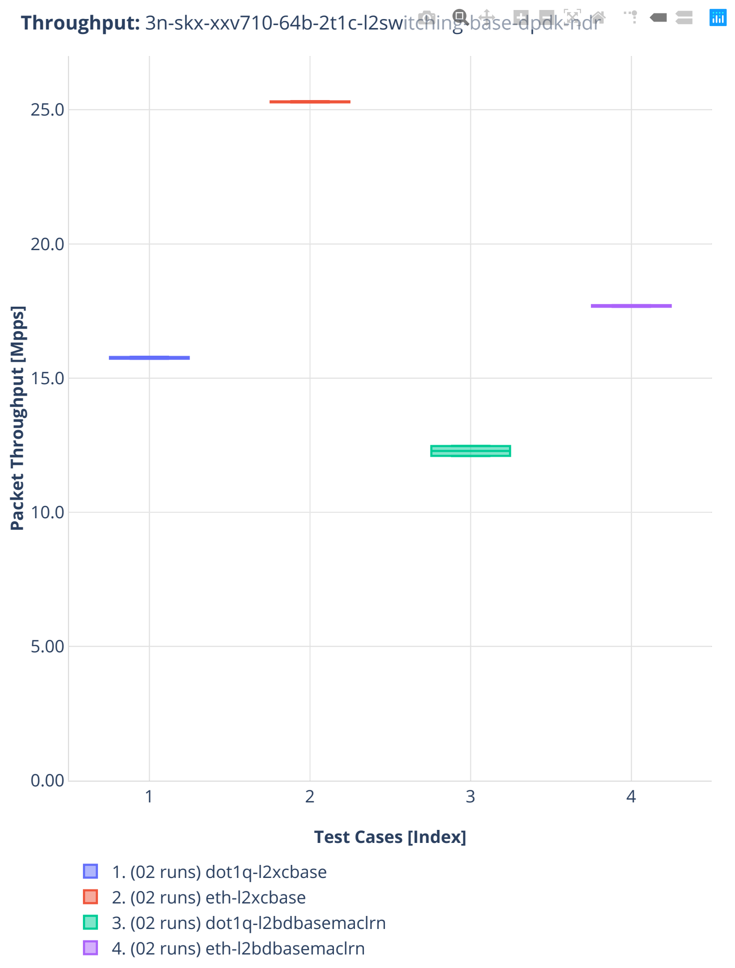
3n-skx-xxv710

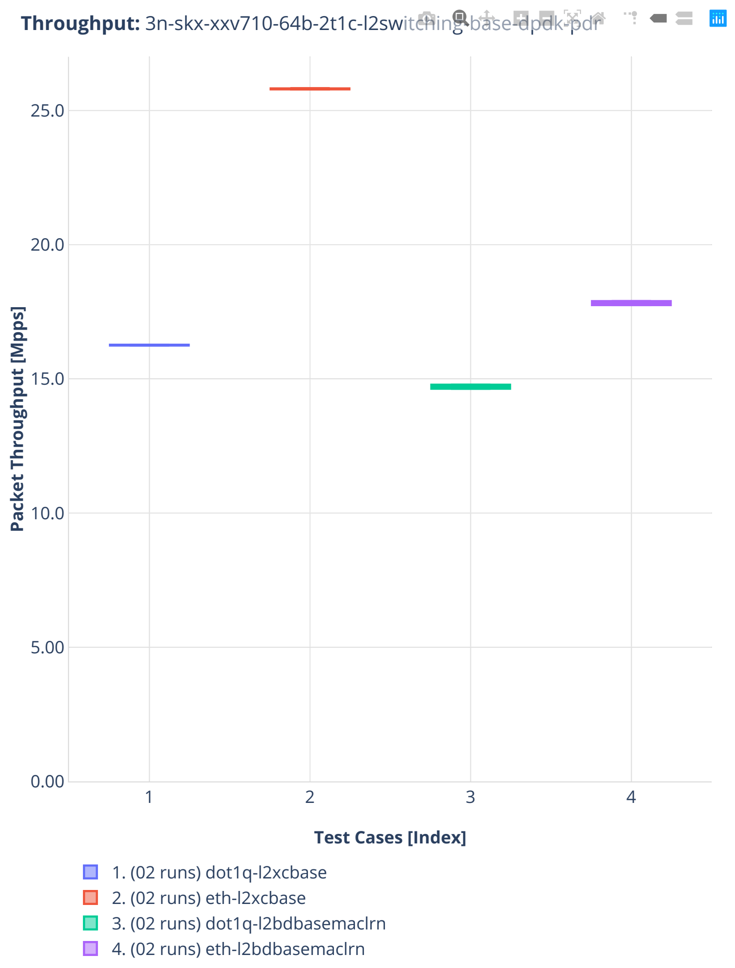
64b-2t1c-l2switching-base-avf



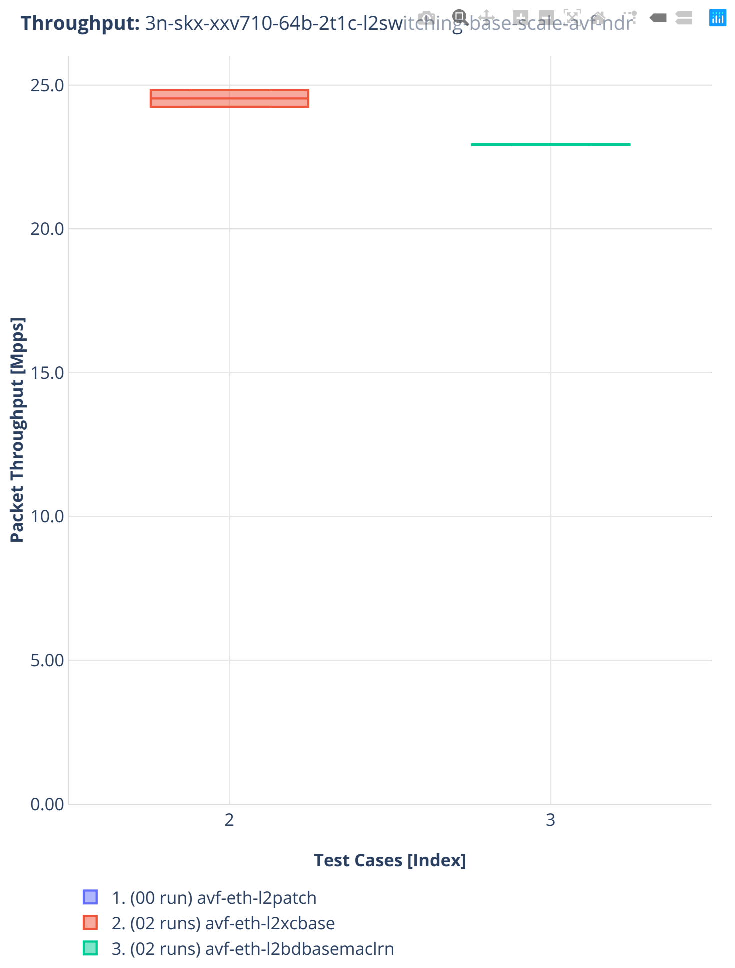

Throughput: 3n-skx-xxv710-64b-2t1c-l2switching-base-avf-pdr
30.0
25.0
20.0
15.0
10.0
5.00
0.00
Packet Throughput [Mpps]
1
2
3
4
Test Cases [Index]
1. (02 runs) avf-dot1q-l2bdbasemaclrn
2. (00 run) avf-eth-l2patch
3. (02 runs) avf-eth-l2xcbase
4. (02 runs) avf-eth-l2bdbasemaclrn

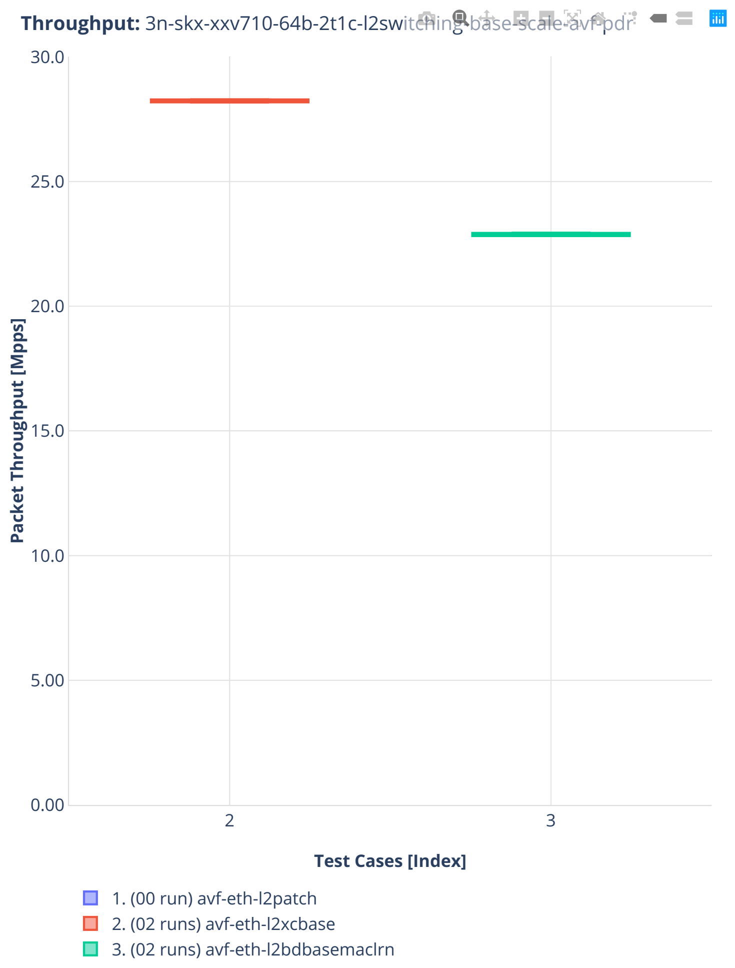
64b-2t1c-l2switching-base-dpdk



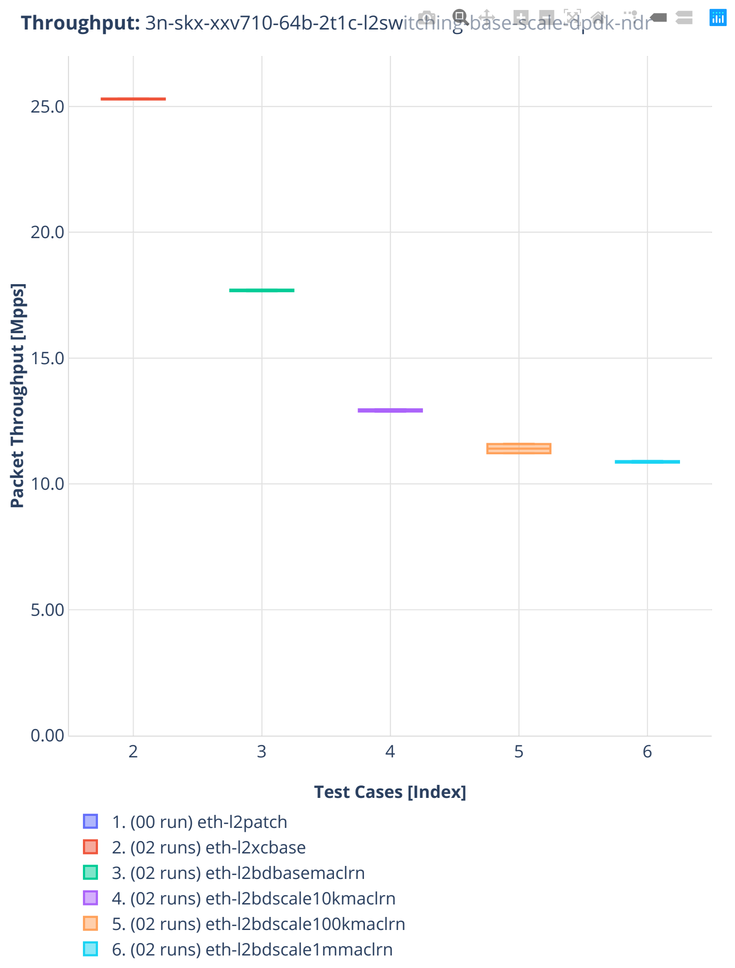

Throughput: 3n-skx-xxv710-64b-2t1c-l2switching-base-dpdk-pdr
Packet Throughput [Mpps]
25.0
20.0
15.0
10.0
5.00
0.00
1
2
3
4
Test Cases [Index]
1. (02 runs) dot1q-l2xcbase
2. (02 runs) eth-l2xcbase
3. (02 runs) dot1q-l2bdbasemaclrn
4. (02 runs) eth-l2bdbasemaclrn

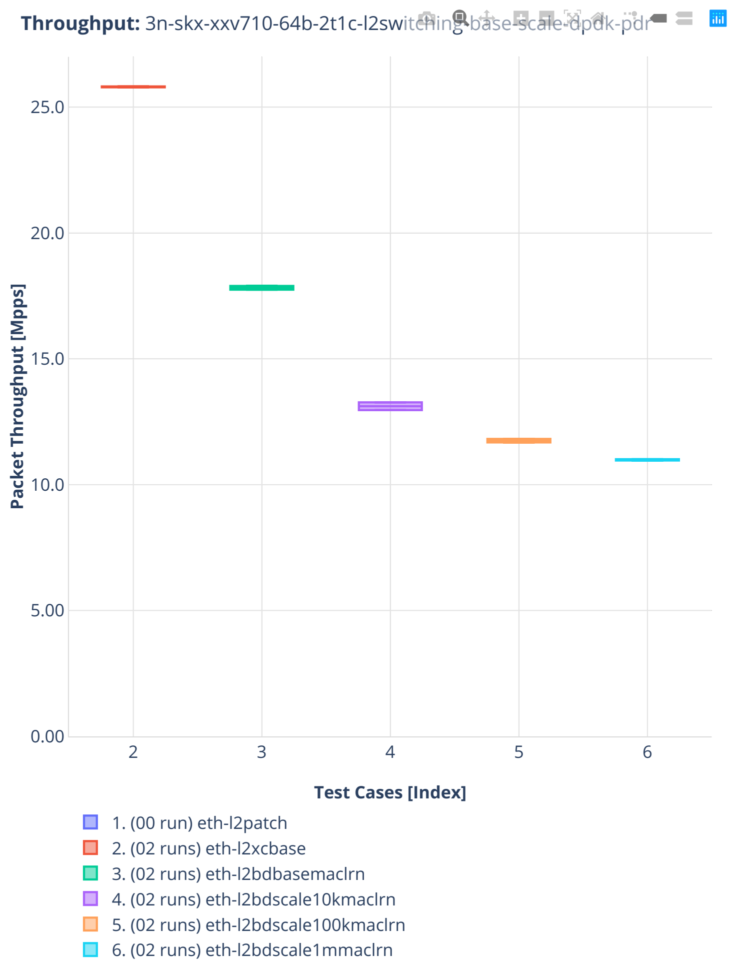
64b-2t1c-l2switching-base-scale-avf



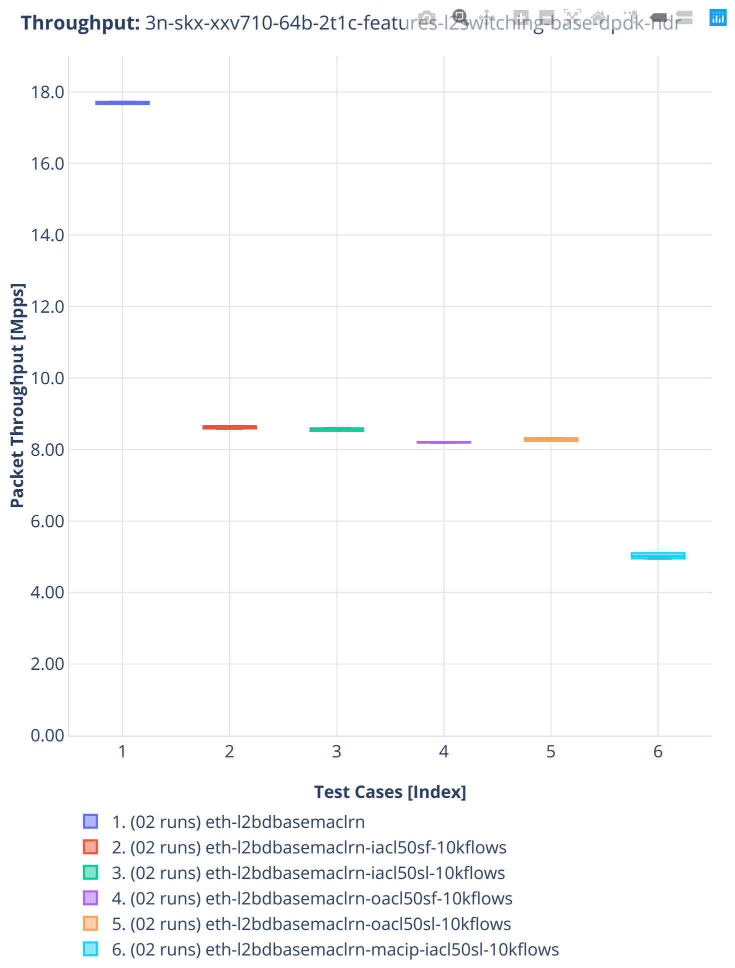

Throughput: 3n-skx-xxv710-64b-2t1c-l2switching-base-scale-avf-pdr
30.0
25.0
20.0
15.0
10.0
5.00
0.00
Packet Throughput [Mpps]
2
3
Test Cases [Index]
1. (00 run) avf-eth-l2patch
2. (02 runs) avf-eth-l2xcbase
3. (02 runs) avf-eth-l2bdbasemaclrn

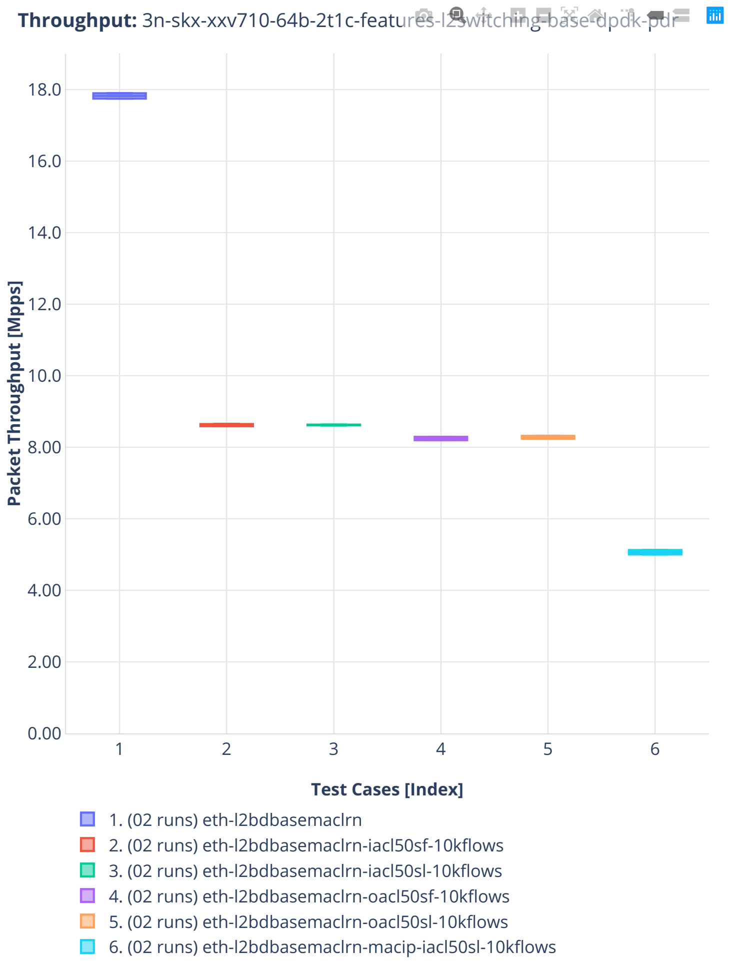
64b-2t1c-l2switching-base-scale-dpdk




Throughput: 3n-skx-xxv710-64b-2t1c-l2switching-base-scale-dpdk-pdr
Packet Throughput [Mpps]
25.0
20.0
15.0
10.0
5.00
0.00
2
3
4
5
6
Test Cases [Index]
1. (00 run) eth-l2patch
2. (02 runs) eth-l2xcbase
3. (02 runs) eth-l2bdbasemaclrn
4. (02 runs) eth-l2bdscale10kmaclrn
5. (02 runs) eth-l2bdscale100kmaclrn
6. (02 runs) eth-l2bdscale1mmaclrn

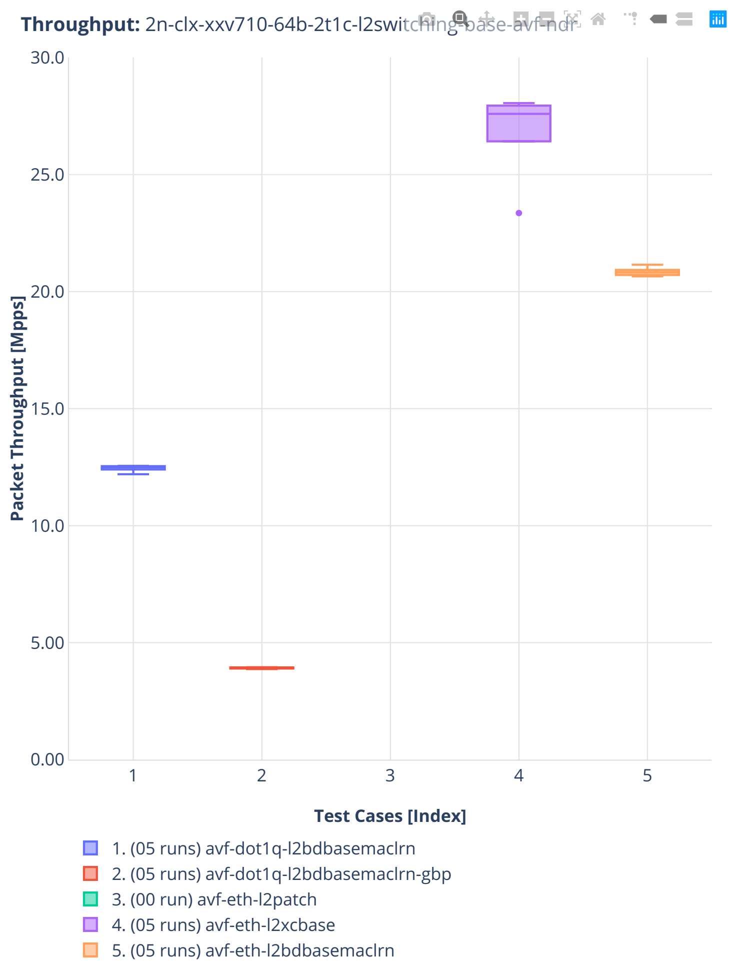
64b-2t1c-features-l2switching-base-dpdk

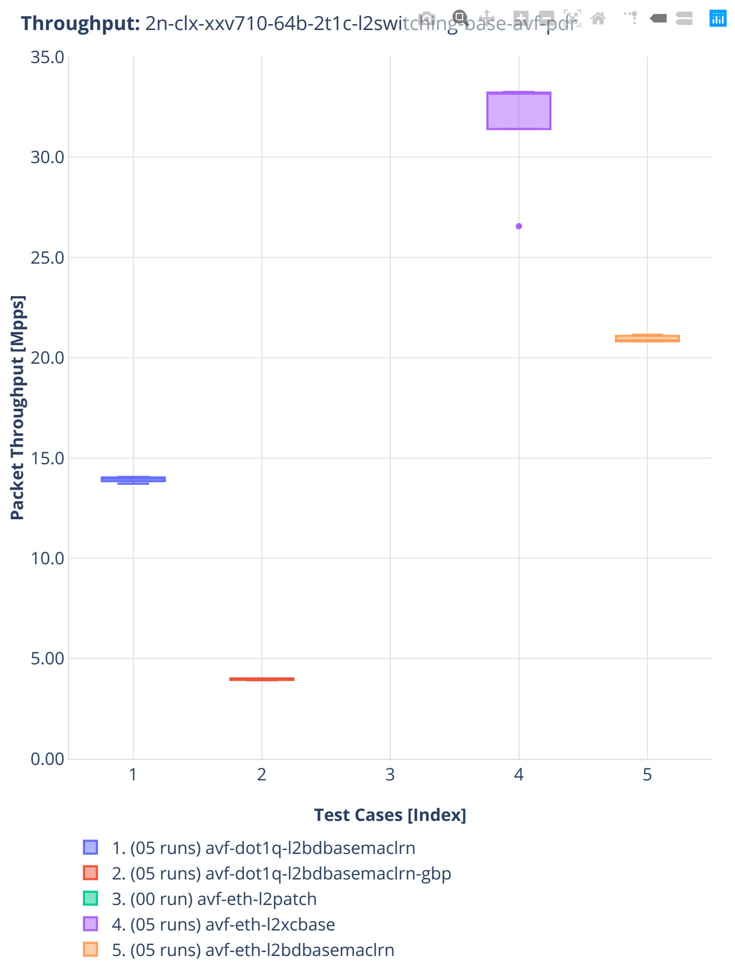



Throughput: 3n-skx-xxv710-64b-2t1c-features-l2switching-base-dpdk-pur
Packet Throughput [Mpps]
18.0
16.0
14.0
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
5
6
Test Cases [Index]
1. (02 runs) eth-l2bdbasemaclrn
2. (02 runs) eth-l2bdbasemaclrn-iacl50sf-10kflows
3. (02 runs) eth-l2bdbasemaclrn-iacl50sl-10kflows
4. (02 runs) eth-l2bdbasemaclrn-oacl50sf-10kflows
5. (02 runs) eth-l2bdbasemaclrn-oacl50sl-10kflows
6. (02 runs) eth-l2bdbasemaclrn-macip-iacl50sl-10kflows

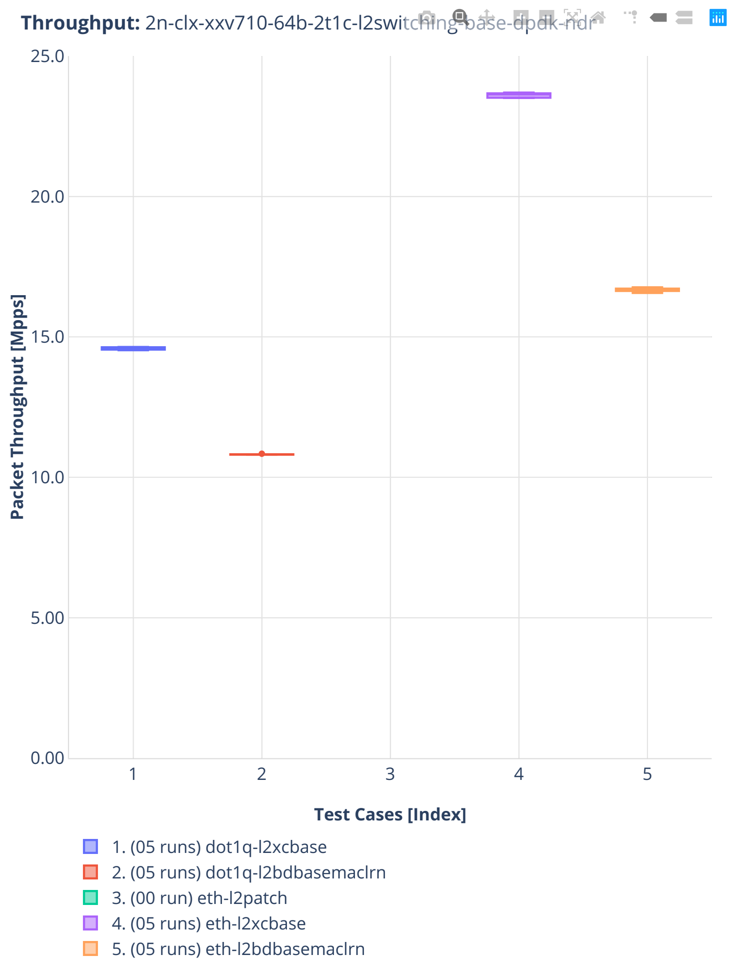
2n-clx-xxv710

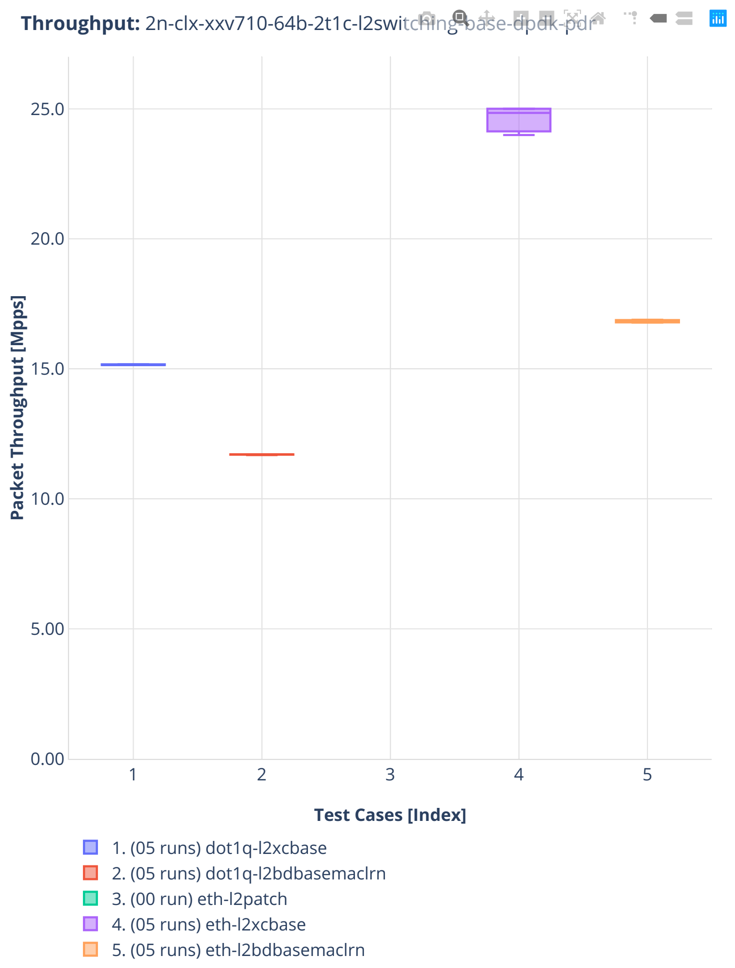
64b-2t1c-l2switching-base-avf



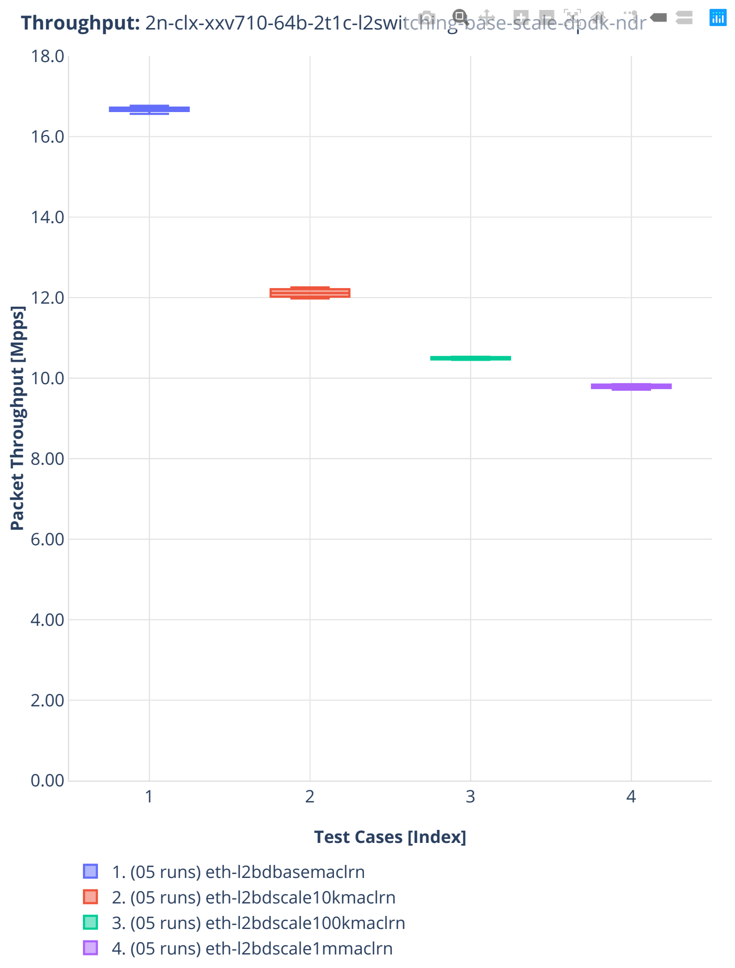

Throughput: 2n-clx-xxv710-64b-2t1c-l2switching-base-avf-pdr
Packet Throughput [Mpps]
35.0
30.0
25.0
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (05 runs) avf-dot1q-l2bdbasemaclrn
2. (05 runs) avf-dot1q-l2bdbasemaclrn-gbp
3. (00 run) avf-eth-l2patch
4. (05 runs) avf-eth-l2xcbase
5. (05 runs) avf-eth-l2bdbasemaclrn

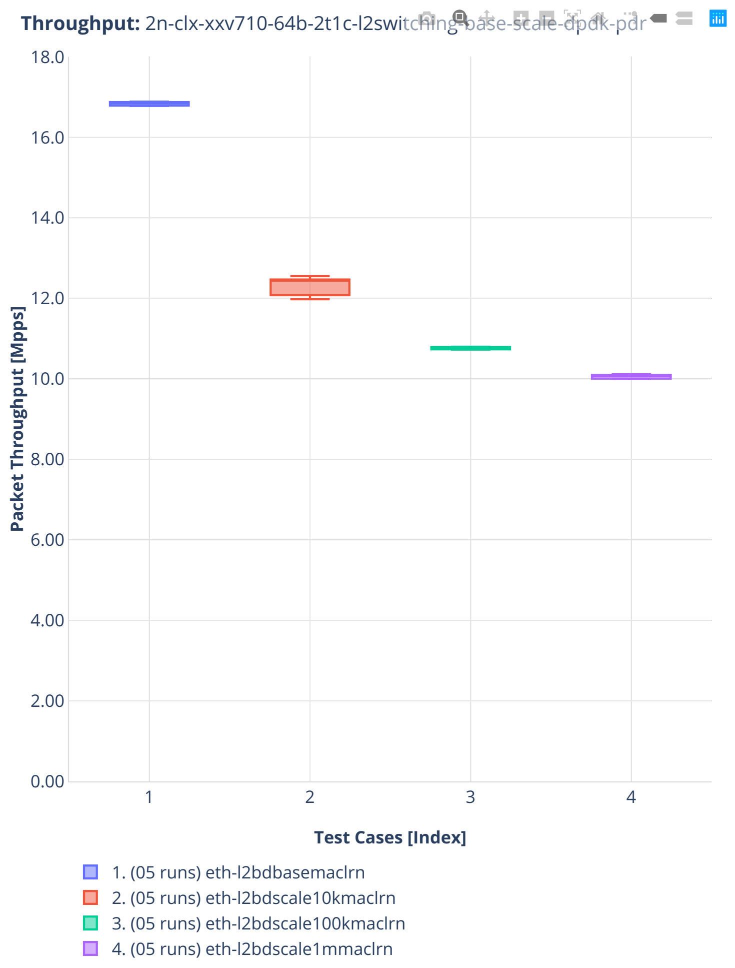
64b-2t1c-l2switching-base-dpdk




Throughput: 2n-clx-xxv710-64b-2t1c-l2switching-base-dpdk-pdr
Packet Throughput [Mpps]
25.0
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
🟦 1. (05 runs) dot1q-l2xcbase
🟥 2. (05 runs) dot1q-l2bdbasemaclrn
🟩 3. (00 run) eth-l2patch
🟪 4. (05 runs) eth-l2xcbase
🟧 5. (05 runs) eth-l2bdbasemaclrn

64b-2t1c-l2switching-base-scale-dpdk




Throughput: 2n-clx-xxv710-64b-2t1c-l2switching-base-scale-dpdk-pdr
Packet Throughput [Mpps]
18.0
16.0
14.0
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
Test Cases [Index]
1. (05 runs) eth-l2bdbasemaclrn
2. (05 runs) eth-l2bdscale10kmaclrn
3. (05 runs) eth-l2bdscale100kmaclrn
4. (05 runs) eth-l2bdscale1mmaclrn

2.3.2 IPv4 Routing

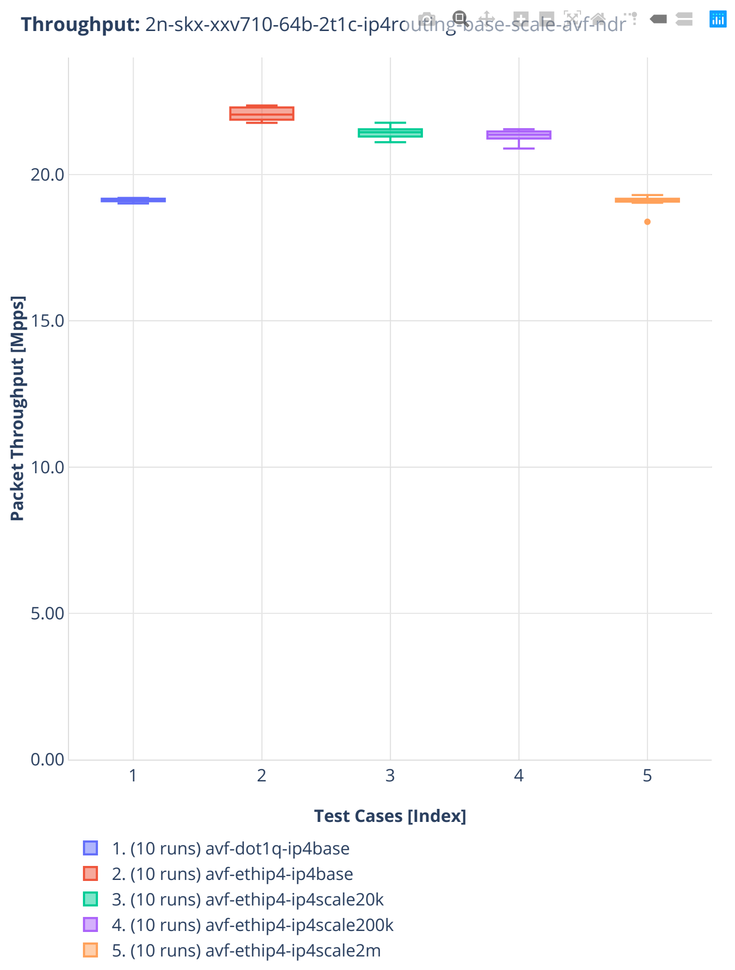
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv4 Routed-Forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

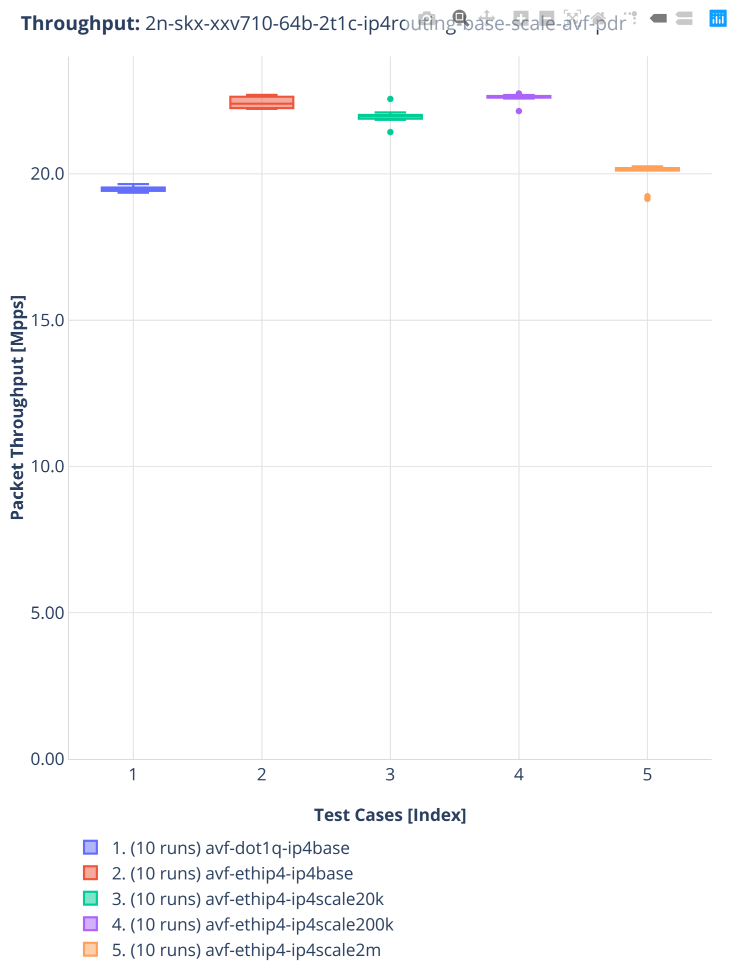
CSIT source code for the test cases used for plots can be found in CSIT git repository[42].

[42] https://git.fd.io/csit/tree/tests/vpp/perf/ip4?h=rls1908_2

2n-skx-xxv710

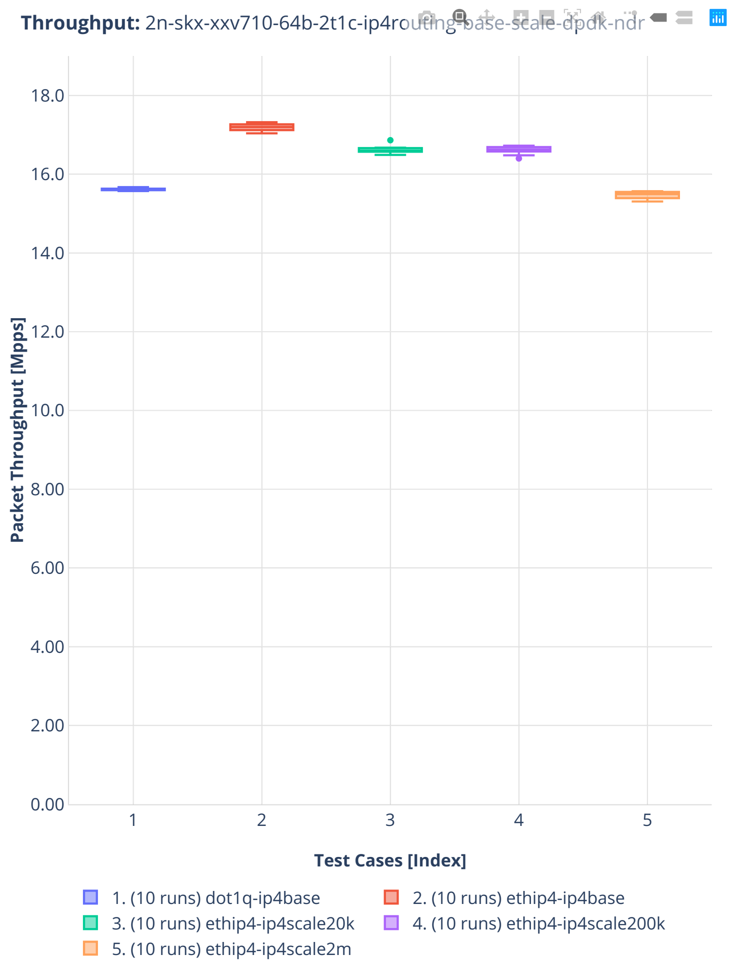
64b-2t1c-ip4routing-base-scale-avf

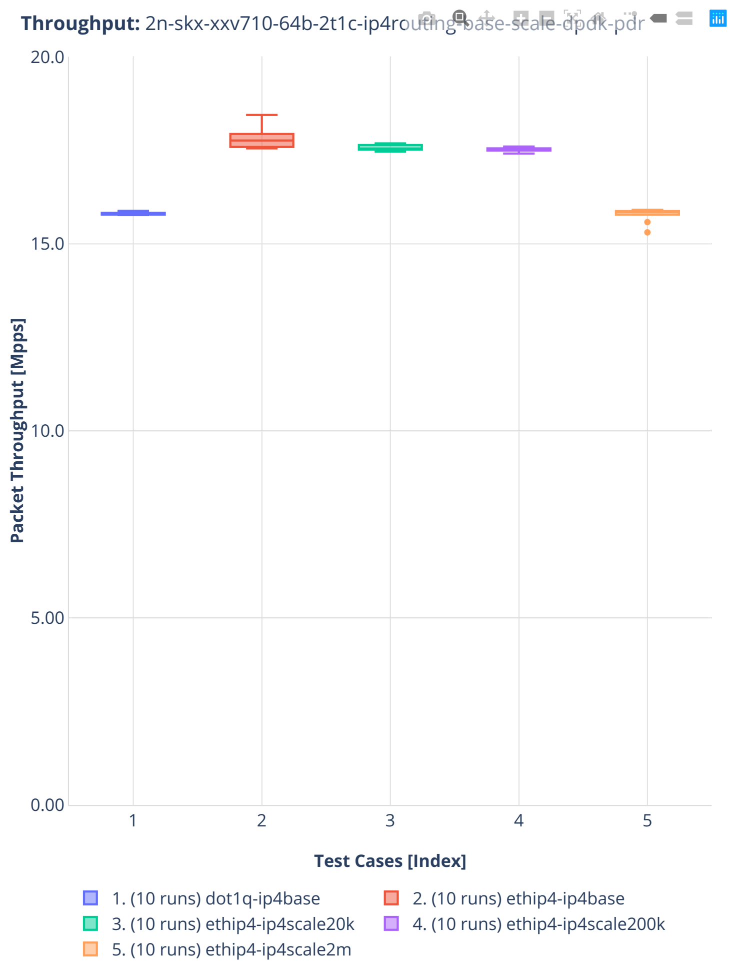




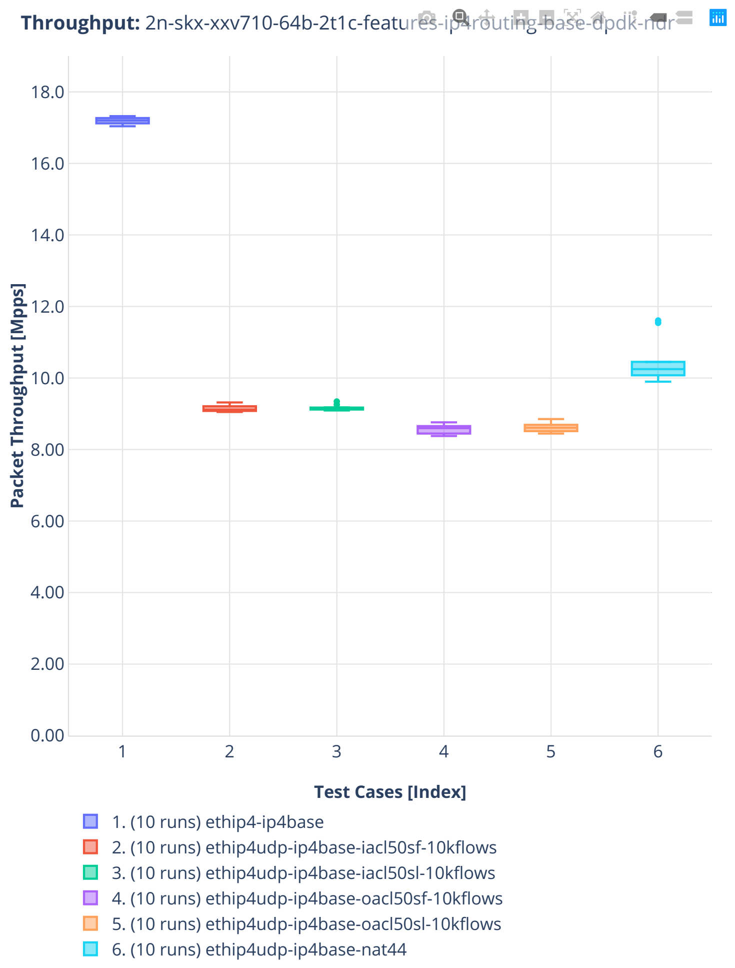
Throughput: 2n-skx-xxv710-64b-2t1c-ip4routing-base-scale-avf-pdr
Packet Throughput [Mpps]
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (10 runs) avf-dot1q-ip4base
2. (10 runs) avf-ethip4-ip4base
3. (10 runs) avf-ethip4-ip4scale20k
4. (10 runs) avf-ethip4-ip4scale200k
5. (10 runs) avf-ethip4-ip4scale2m

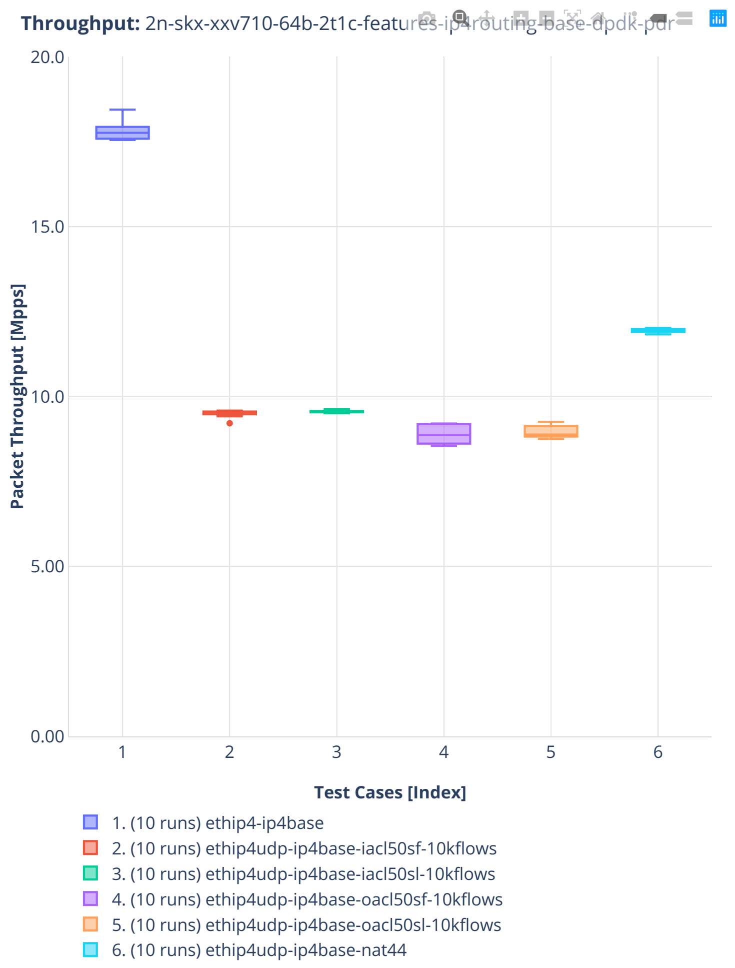
64b-2t1c-ip4routing-base-scale-dpdk




Throughput: 2n-skx-xxv710-64b-2t1c-ip4routing-base-scale-dpdk-pdr
20.0
15.0
10.0
5.00
0.00
Packet Throughput [Mpps]
1
2
3
4
5
Test Cases [Index]
1. (10 runs) dot1q-ip4base
2. (10 runs) ethip4-ip4base
3. (10 runs) ethip4-ip4scale20k
4. (10 runs) ethip4-ip4scale200k
5. (10 runs) ethip4-ip4scale2m

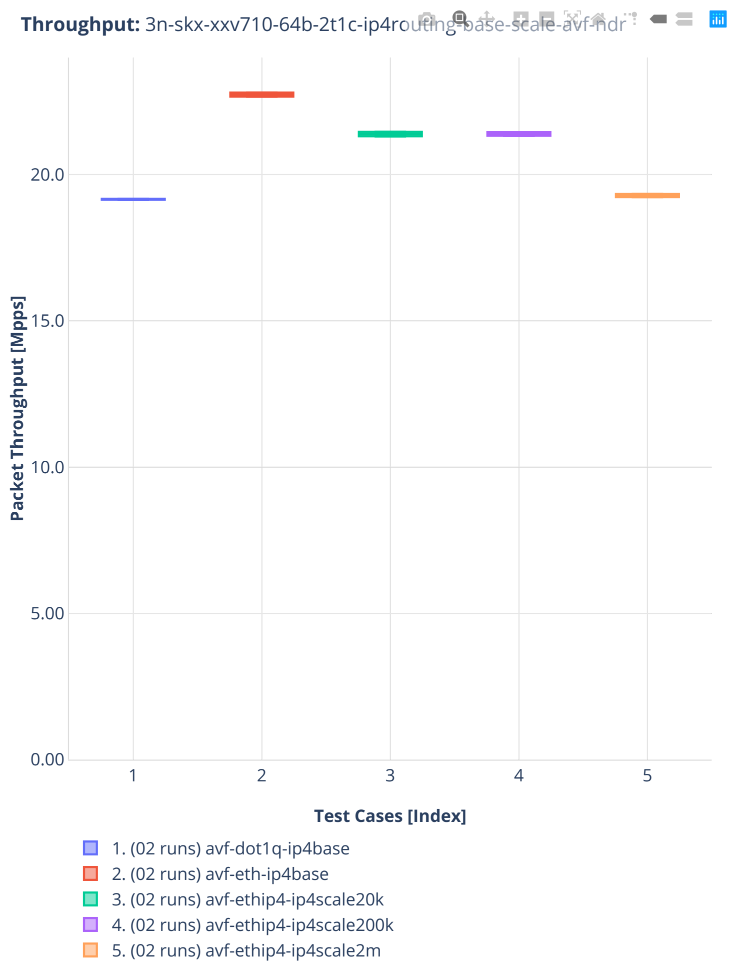
64b-2t1c-features-ip4routing-base-dpdk

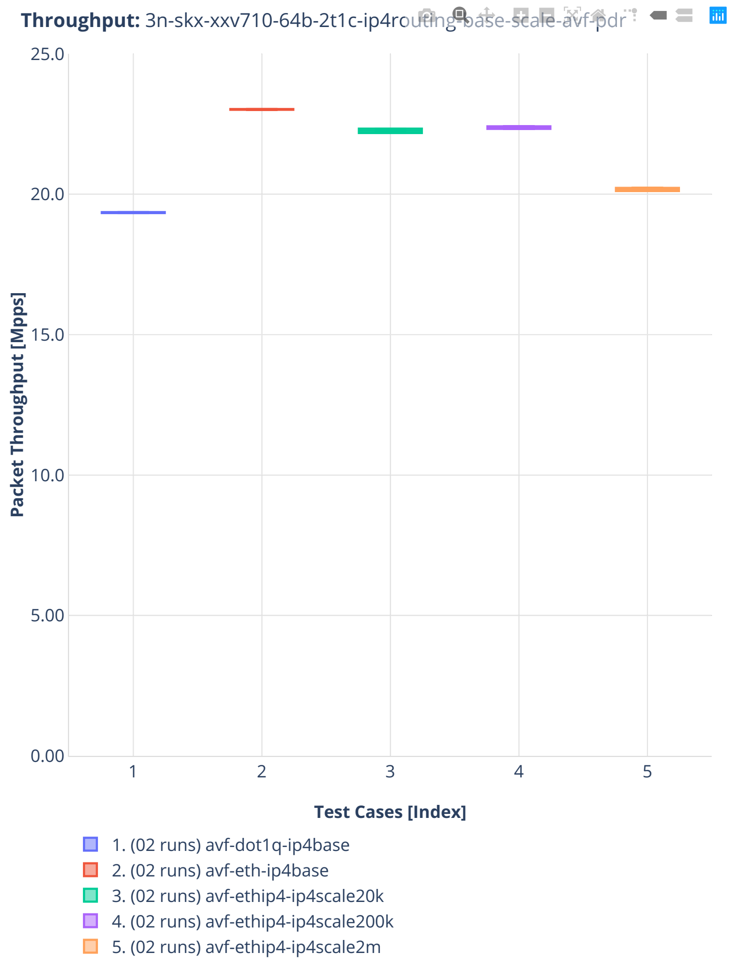



Throughput: 2n-skx-xxv710-64b-2t1c-features-ip4routing-base-dpdk-pdr
20.0
15.0
10.0
5.00
0.00
Packet Throughput [Mpps]
1
2
3
4
5
6
Test Cases [Index]
1. (10 runs) ethip4-ip4base
2. (10 runs) ethip4udp-ip4base-iacl50sf-10kflows
3. (10 runs) ethip4udp-ip4base-iacl50sl-10kflows
4. (10 runs) ethip4udp-ip4base-oacl50sf-10kflows
5. (10 runs) ethip4udp-ip4base-oacl50sl-10kflows
6. (10 runs) ethip4udp-ip4base-nat44

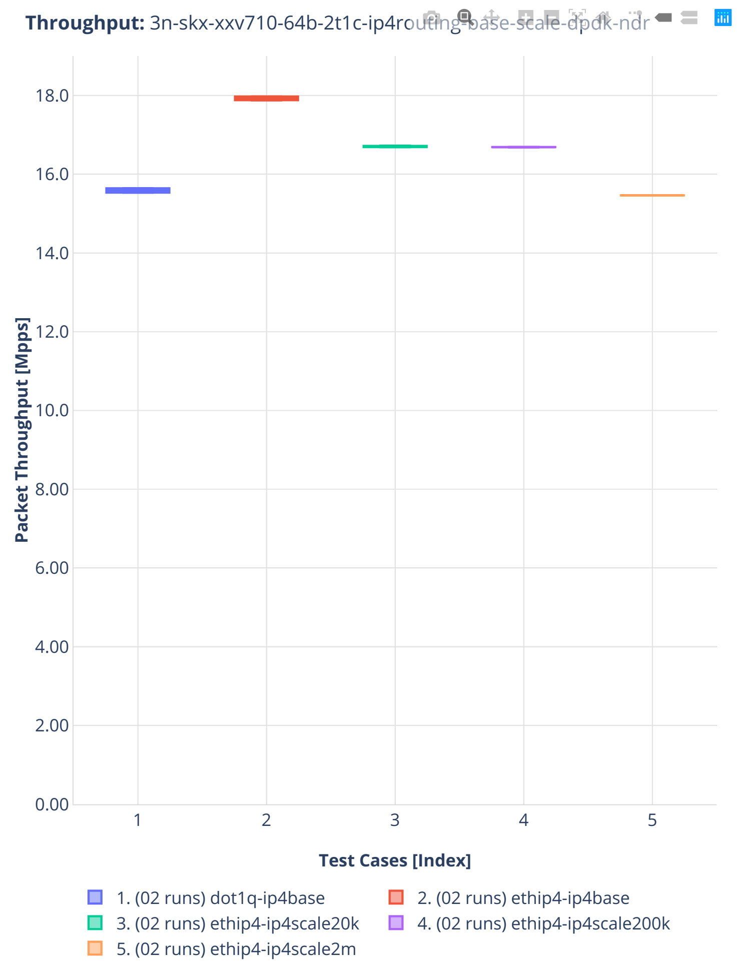
3n-skx-xxv710

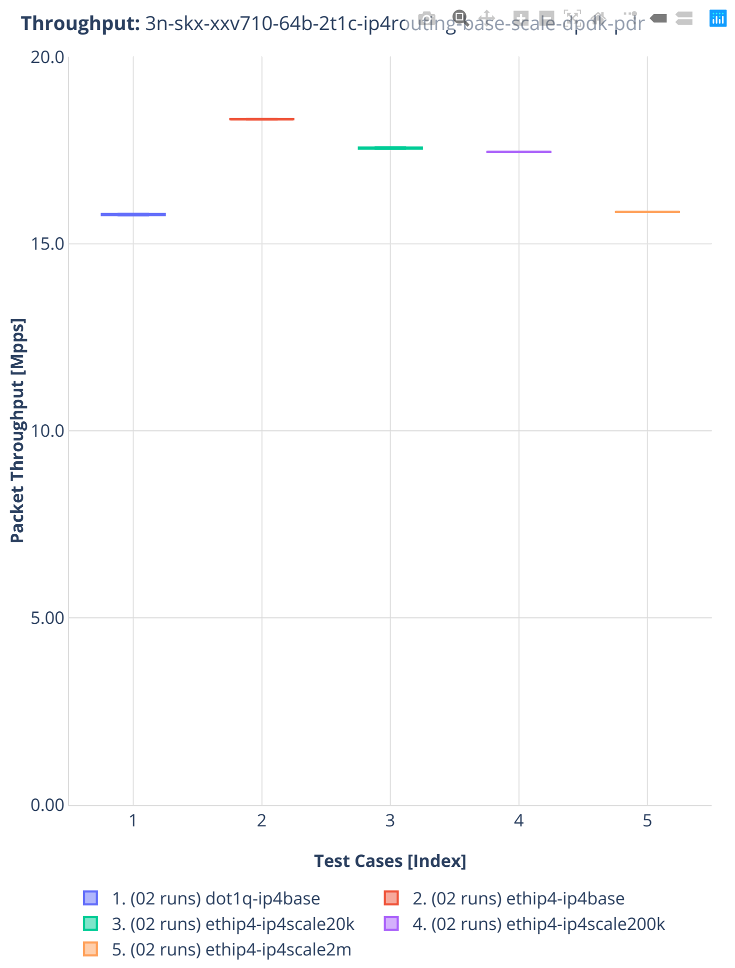
64b-2t1c-ip4routing-base-scale-avf



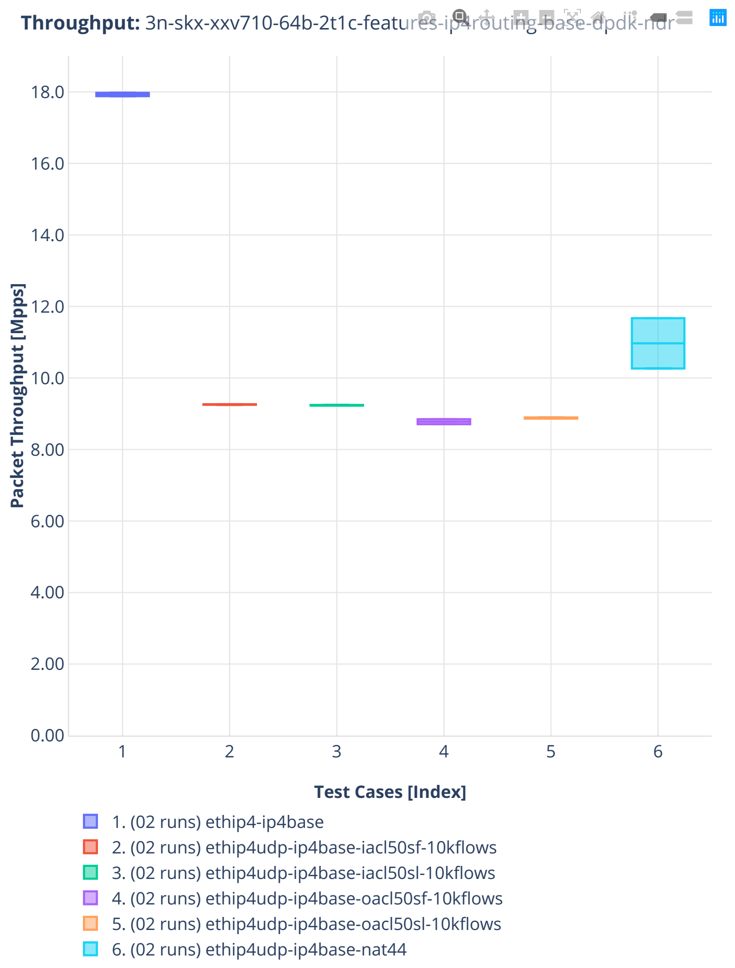

Throughput: 3n-skx-xxv710-64b-2t1c-ip4routing-base-scale-avf-pdr
25.0
20.0
15.0
10.0
5.00
0.00
Packet Throughput [Mpps]
1
2
3
4
5
Test Cases [Index]
1. (02 runs) avf-dot1q-ip4base
2. (02 runs) avf-eth-ip4base
3. (02 runs) avf-ethip4-ip4scale20k
4. (02 runs) avf-ethip4-ip4scale200k
5. (02 runs) avf-ethip4-ip4scale2m

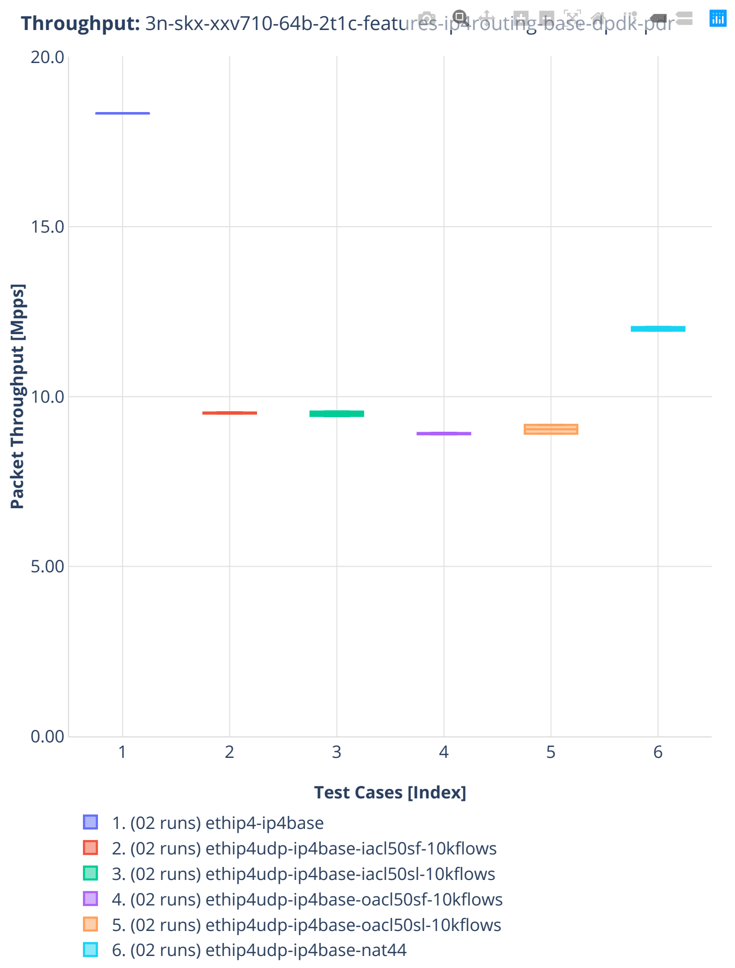
64b-2t1c-ip4routing-base-scale-dpdk




Throughput: 3n-skx-xxv710-64b-2t1c-ip4routing-base-scale-dpdk-pdr
Packet Throughput [Mpps]
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (02 runs) dot1q-ip4base
2. (02 runs) ethip4-ip4base
3. (02 runs) ethip4-ip4scale20k
4. (02 runs) ethip4-ip4scale200k
5. (02 runs) ethip4-ip4scale2m

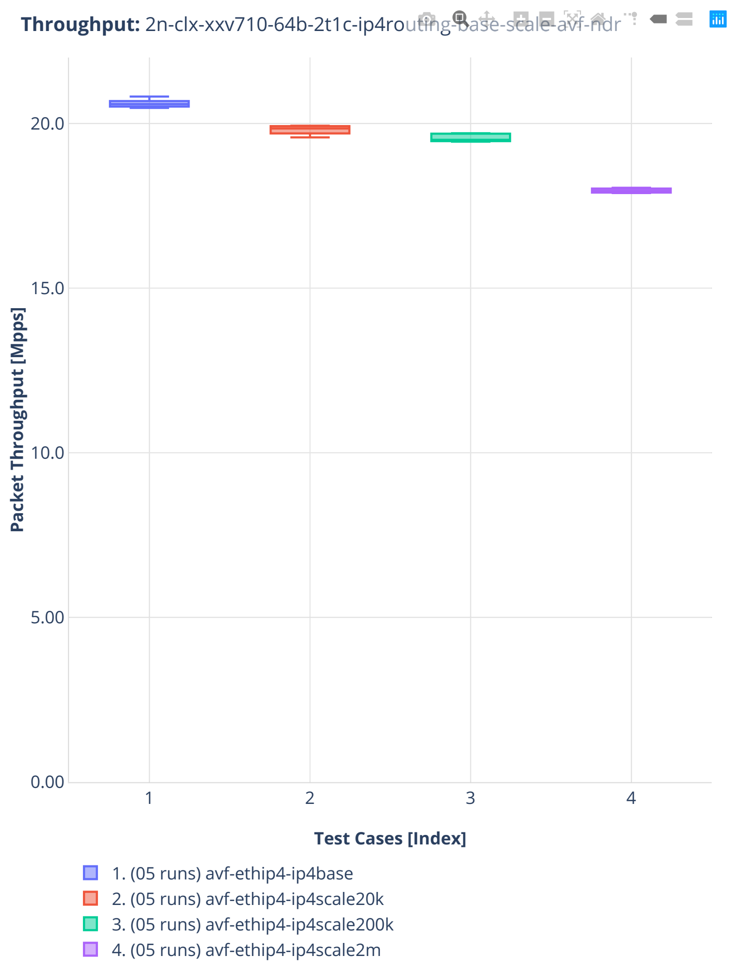
64b-2t1c-features-ip4routing-base-dpdk

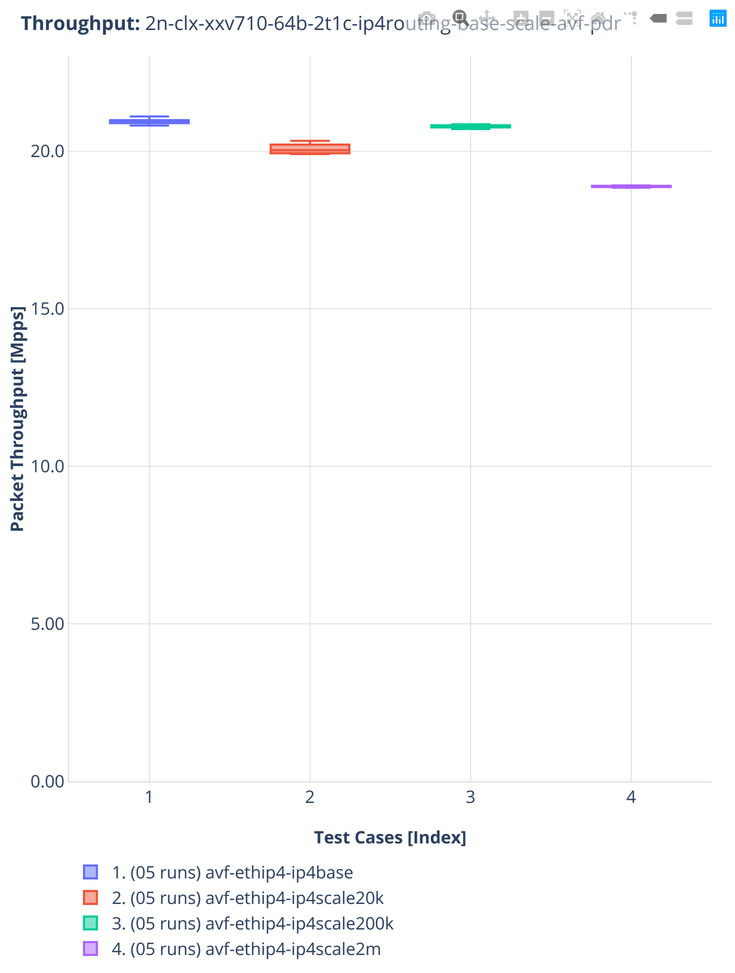



Throughput: 3n-skx-xxv710-64b-2t1c-features-ip4routing-base-dpdk-pdr
Packet Throughput [Mpps]
20.0
15.0
10.0
5.00
0.00
1
2
3
4
5
6
Test Cases [Index]
1. (02 runs) ethip4-ip4base
2. (02 runs) ethip4udp-ip4base-iacl50sf-10kflows
3. (02 runs) ethip4udp-ip4base-iacl50sl-10kflows
4. (02 runs) ethip4udp-ip4base-oacl50sf-10kflows
5. (02 runs) ethip4udp-ip4base-oacl50sl-10kflows
6. (02 runs) ethip4udp-ip4base-nat44

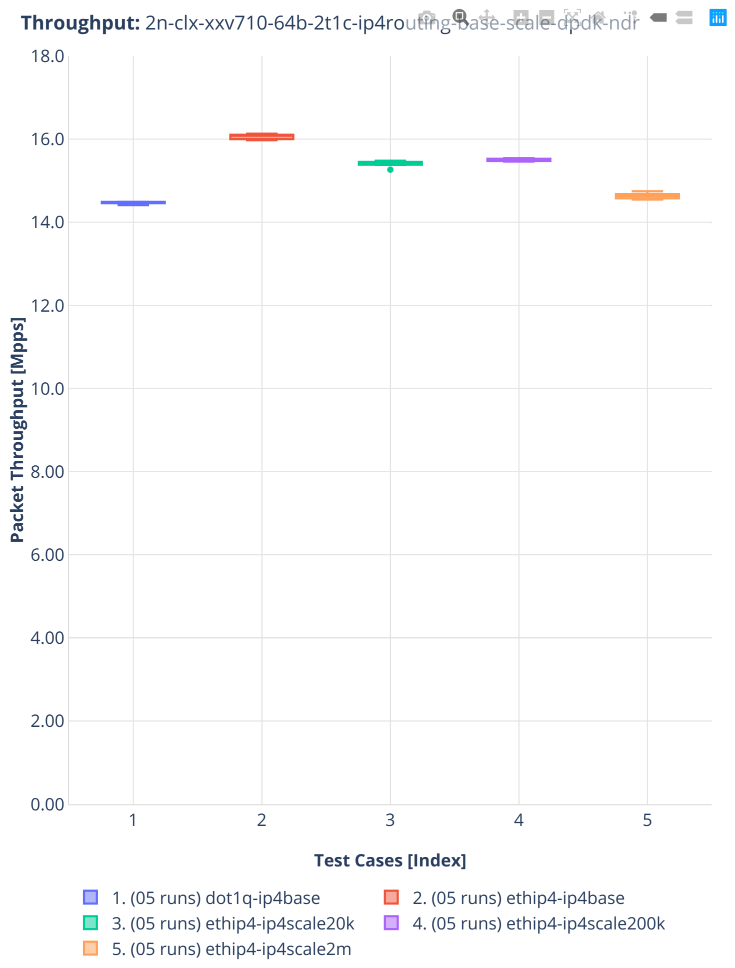
2n-clx-xxv710

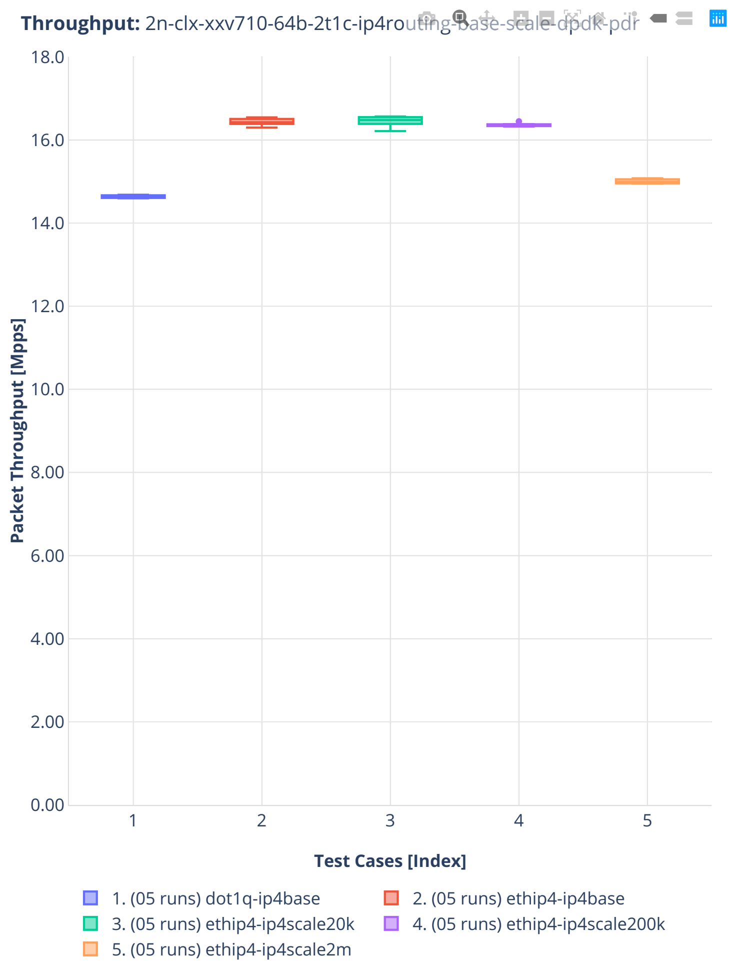
64b-2t1c-ip4routing-base-scale-avf



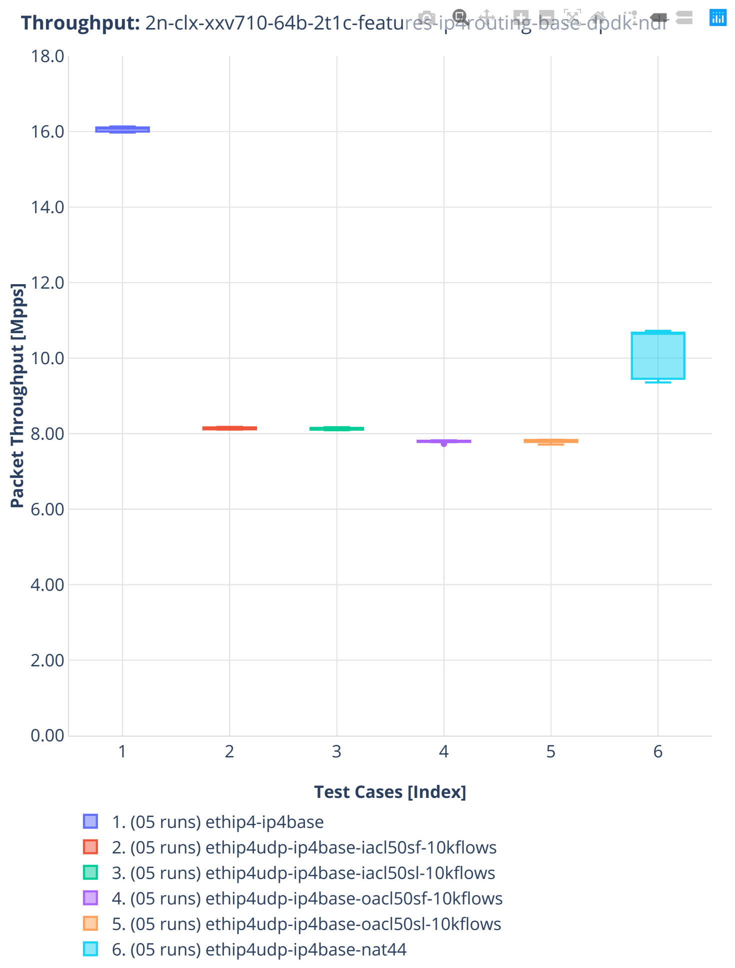

Throughput: 2n-clx-xxv710-64b-2t1c-ip4routing-base-scale-avf-pdr
Packet Throughput [Mpps]
20.0
15.0
10.0
5.00
0.00
1
2
3
4
Test Cases [Index]
1. (05 runs) avf-ethip4-ip4base
2. (05 runs) avf-ethip4-ip4scale20k
3. (05 runs) avf-ethip4-ip4scale200k
4. (05 runs) avf-ethip4-ip4scale2m

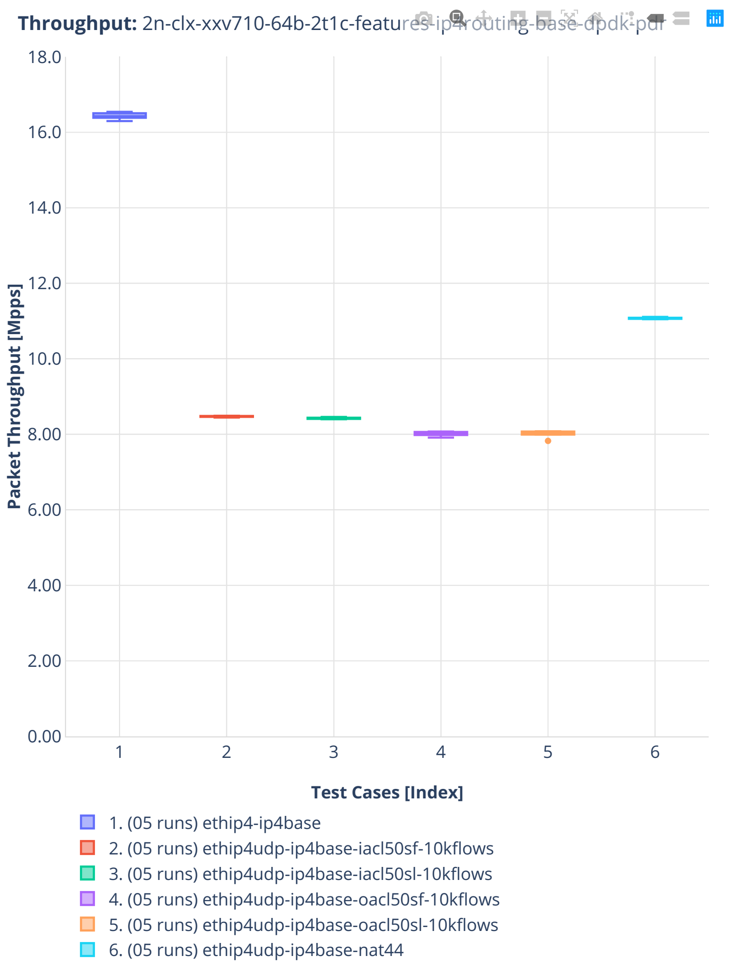
64b-2t1c-ip4routing-base-scale-dpdk




Throughput: 2n-clx-xxv710-64b-2t1c-ip4routing-base-scale-dpdk-pdr
Packet Throughput [Mpps]
18.0
16.0
14.0
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (05 runs) dot1q-ip4base
2. (05 runs) ethip4-ip4base
3. (05 runs) ethip4-ip4scale20k
4. (05 runs) ethip4-ip4scale200k
5. (05 runs) ethip4-ip4scale2m

64b-2t1c-features-ip4routing-base-dpdk





Throughput: 2n-clx-xxv710-64b-2t1c-features-ip4routing-base-dpdk-pdr
Packet Throughput [Mpps]
18.0
16.0
14.0
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
5
6
Test Cases [Index]
1. (05 runs) ethip4-ip4base
2. (05 runs) ethip4udp-ip4base-iacl50sf-10kflows
3. (05 runs) ethip4udp-ip4base-iacl50sl-10kflows
4. (05 runs) ethip4udp-ip4base-oacl50sf-10kflows
5. (05 runs) ethip4udp-ip4base-oacl50sl-10kflows
6. (05 runs) ethip4udp-ip4base-nat44

2.3.3 IPv4 Tunnels

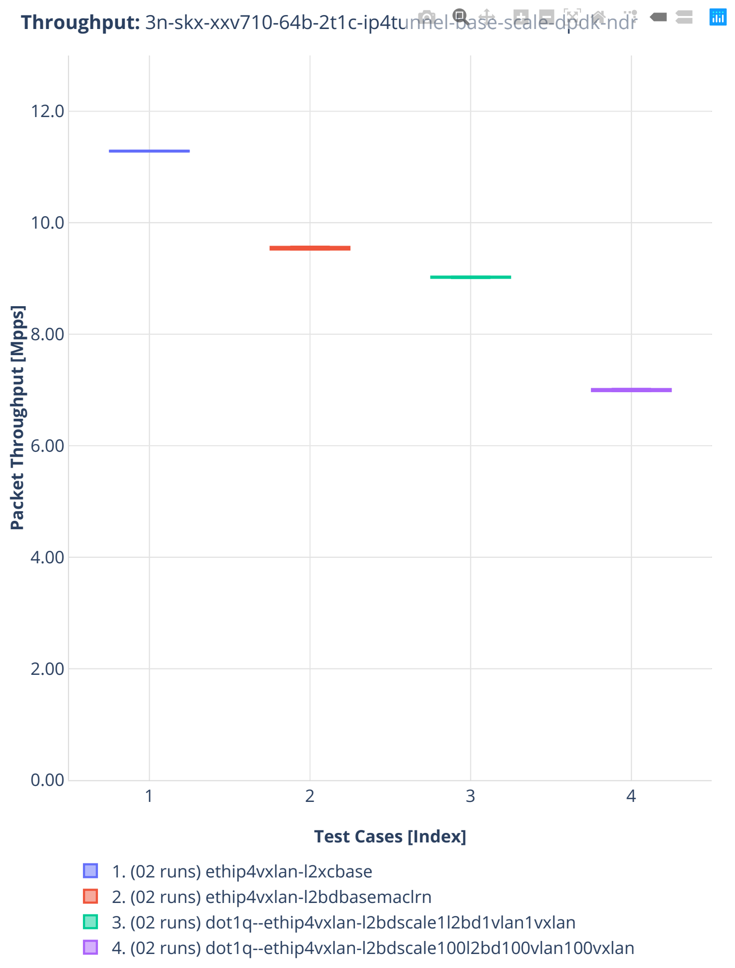
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv4 Overlay Tunnels, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

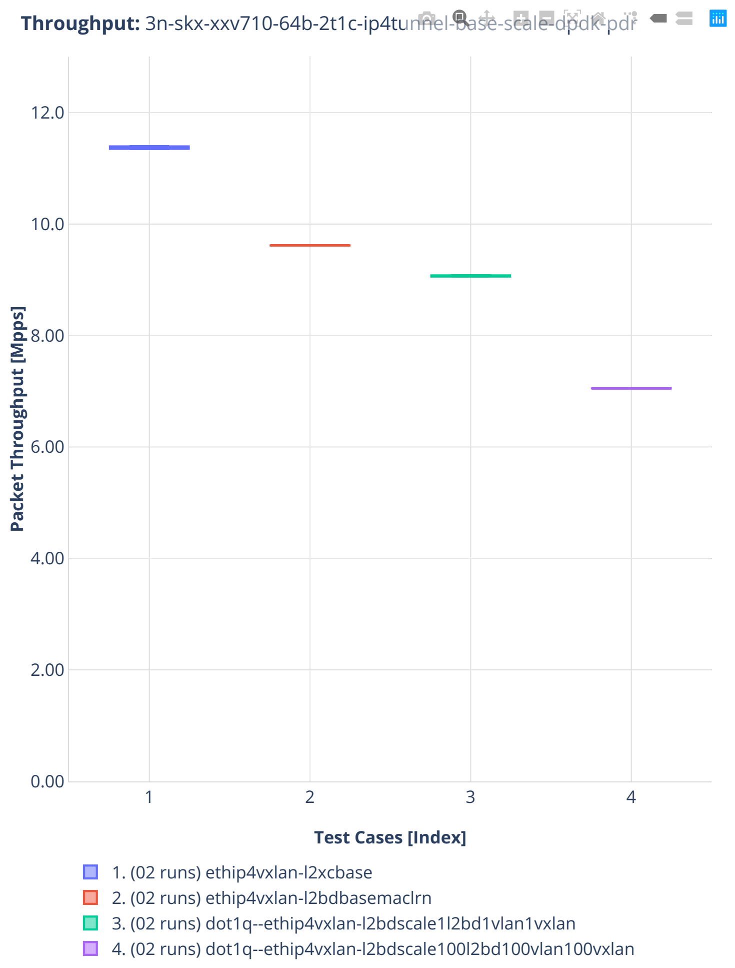
CSIT source code for the test cases used for plots can be found in CSIT git repository[43].

[43] https://git.fd.io/csit/tree/tests/vpp/perf/ip4_tunnels?h=rls1908_2

3n-skx-xxv710

64b-2t1c-ip4tunnel-base-scale-dpdk




Throughput: 3n-skx-xxv710-64b-2t1c-ip4tunnel-base-scale-dpdk-pdr
Packet Throughput [Mpps]
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
Test Cases [Index]
1. (02 runs) ethip4vxlan-l2xcbase
2. (02 runs) ethip4vxlan-l2bdbasemaclrn
3. (02 runs) dot1q--ethip4vxlan-l2bdscale1l2bd1vlan1vxlan
4. (02 runs) dot1q--ethip4vxlan-l2bdscale100l2bd100vlan100vxlan

2.3.4 KVM VMs vhost-user

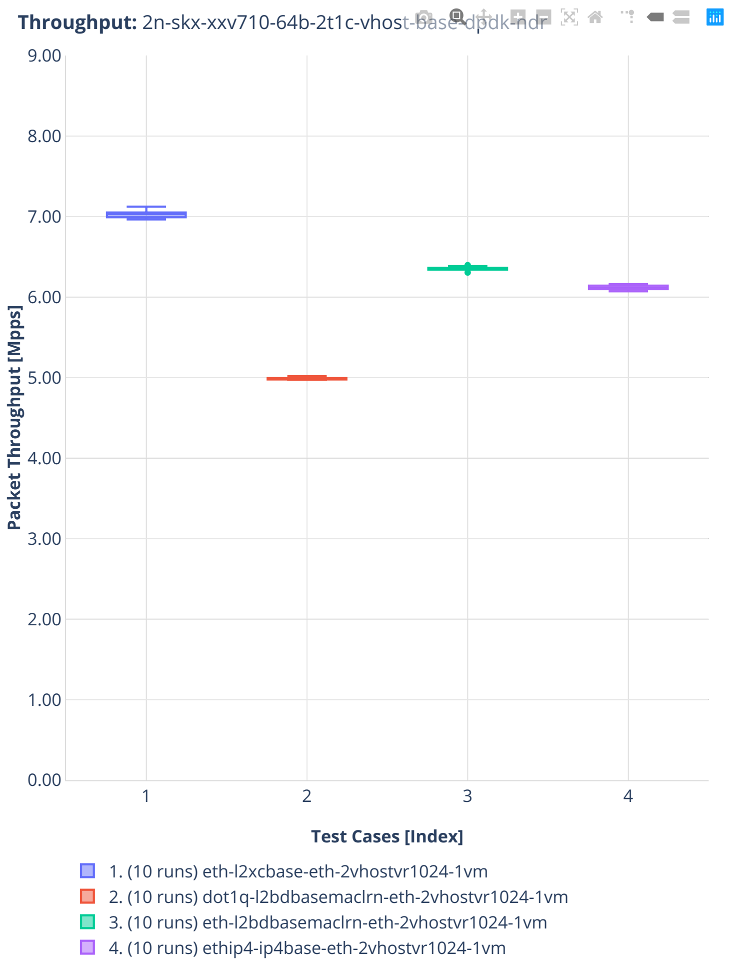
Following sections include summary graphs of VPP Phy-to-VM(s)-to-Phy performance with VM virtio and VPP vhost-user virtual interfaces, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

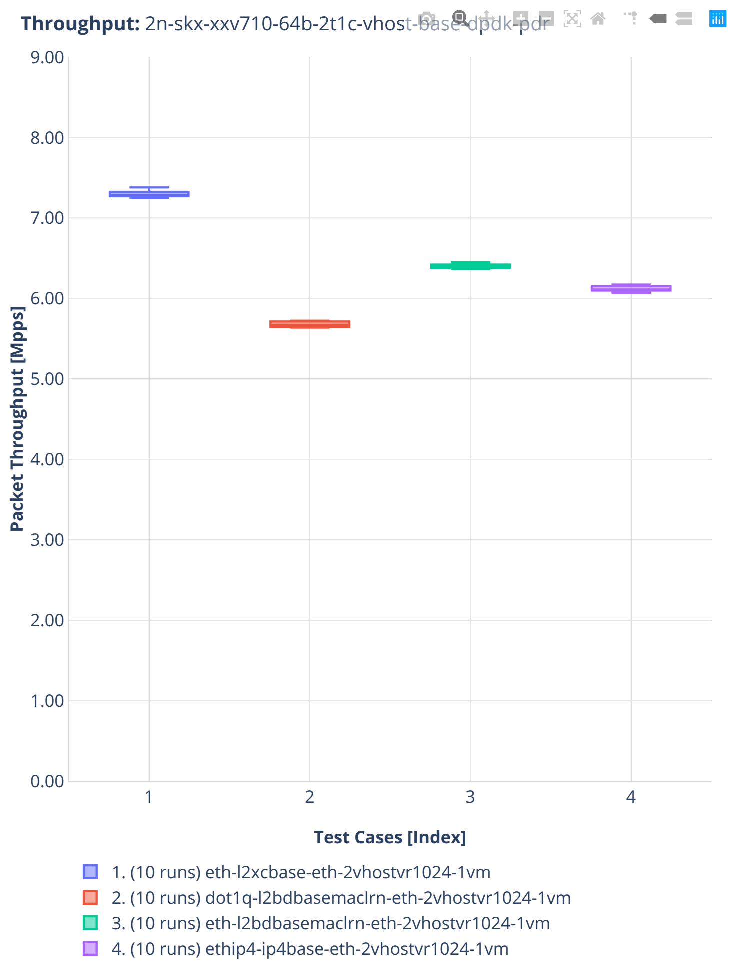
CSIT source code for the test cases used for plots can be found in CSIT git repository[44].

[44] https://git.fd.io/csit/tree/tests/vpp/perf/vm_vhost?h=rls1908_2

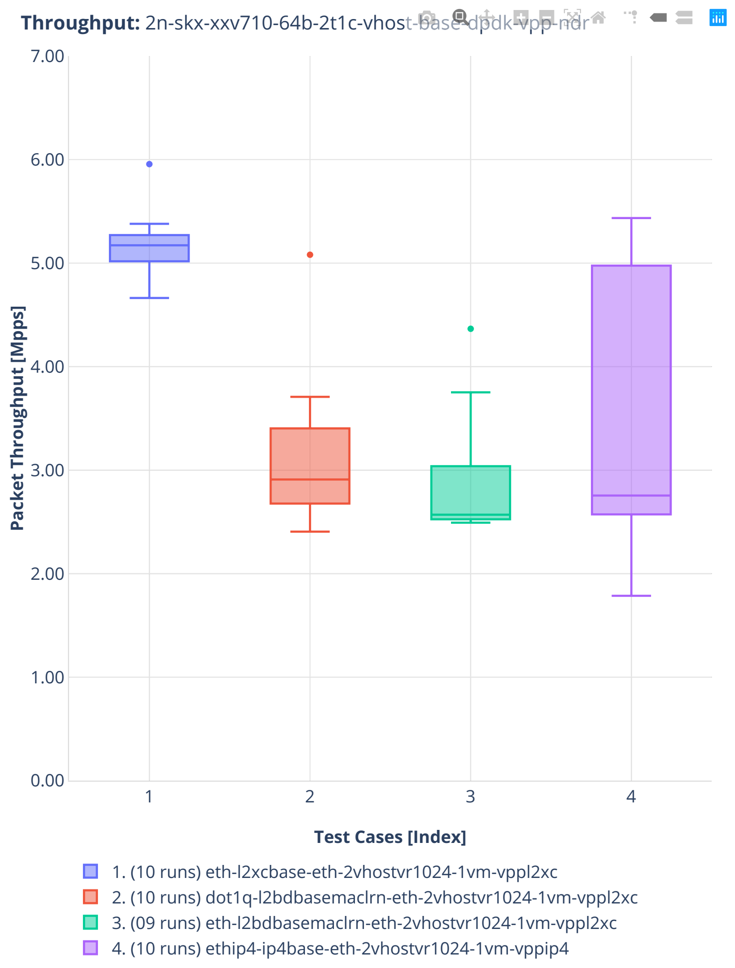
2n-skx-xxv710

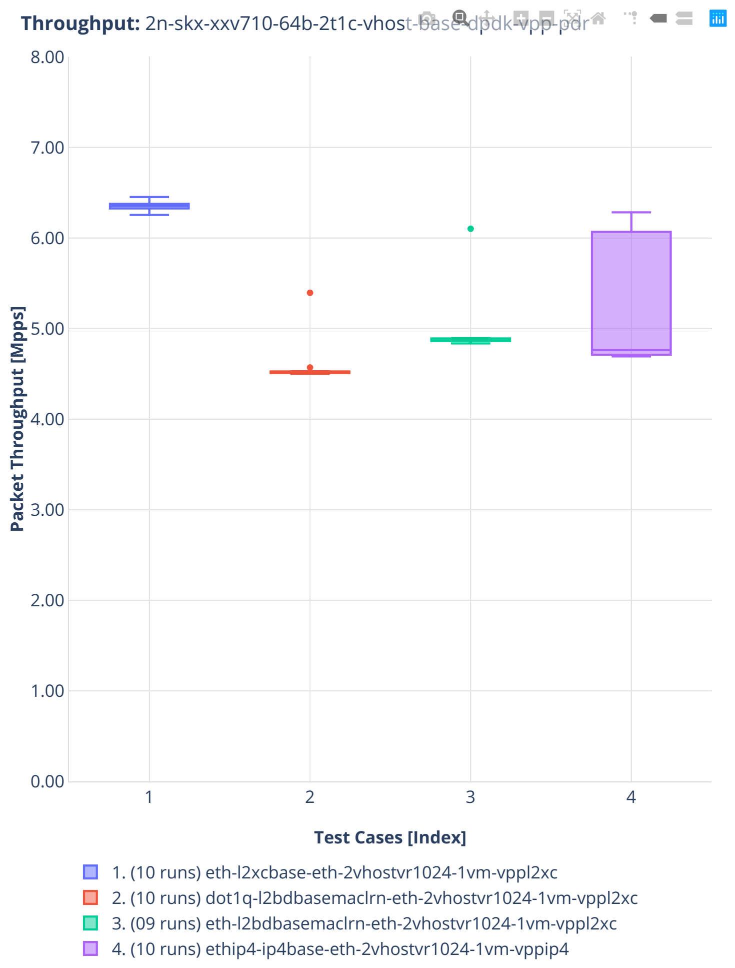
64b-2t1c-vhost-base-dpdk-testpmd




Throughput: 2n-skx-xxv710-64b-2t1c-vhost-base-dpdk-pdr
Packet Throughput [Mpps]
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1
2
3
4
Test Cases [Index]
1. (10 runs) eth-l2xcbase-eth-2vhostvr1024-1vm
2. (10 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm
3. (10 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm
4. (10 runs) ethip4-ip4base-eth-2vhostvr1024-1vm

64b-2t1c-vhost-base-dpdk-vpp

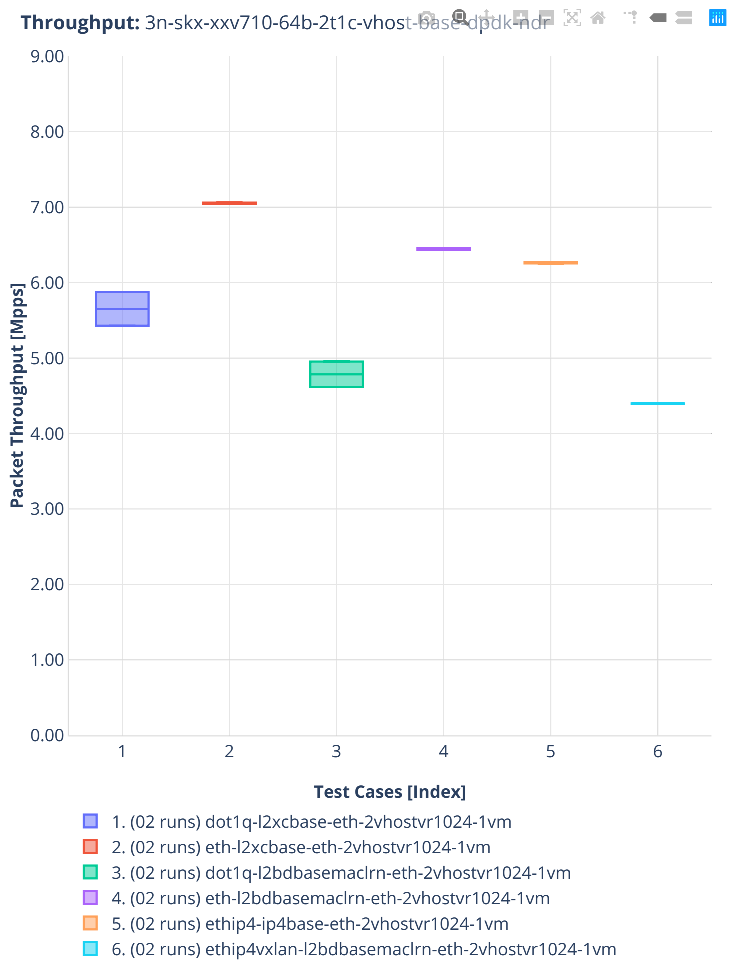


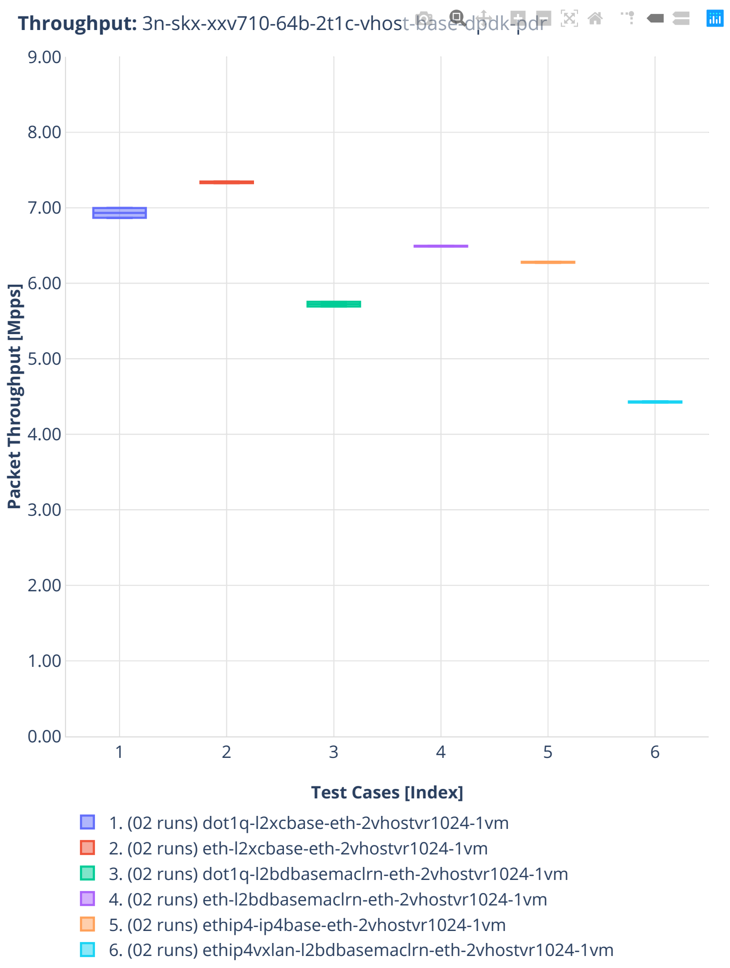

Throughput: 2n-skx-xxv710-64b-2t1c-vhost-base-dpdk-vpp-pdr
Packet Throughput [Mpps]
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1
2
3
4
Test Cases [Index]
1. (10 runs) eth-l2xcbase-eth-2vhostvr1024-1vm-vppl2xc
2. (10 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
3. (09 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
4. (10 runs) ethip4-ip4base-eth-2vhostvr1024-1vm-vppip4

3n-skx-xxv710

64b-2t1c-vhost-base-avf-testpmd

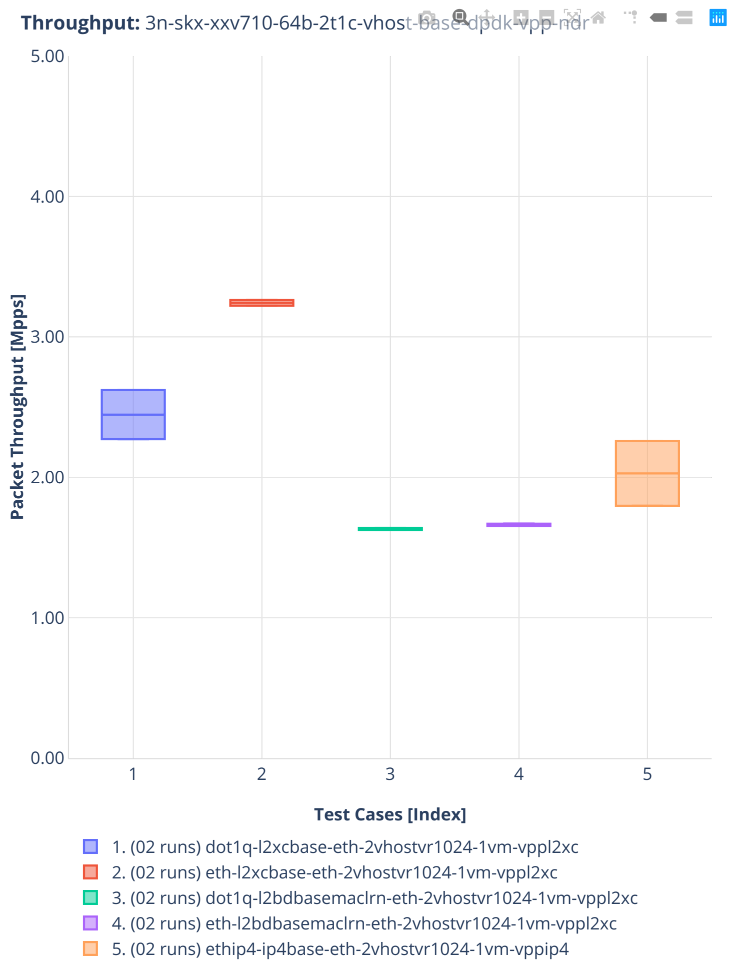
64b-2t1c-vhost-base-dpdk-testpmd

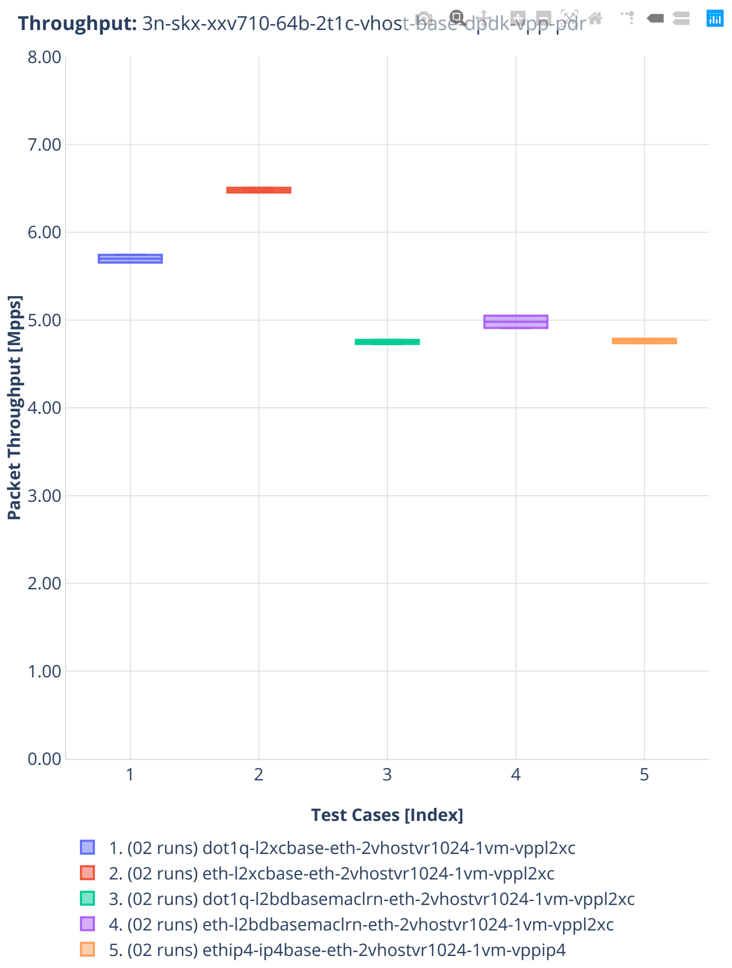




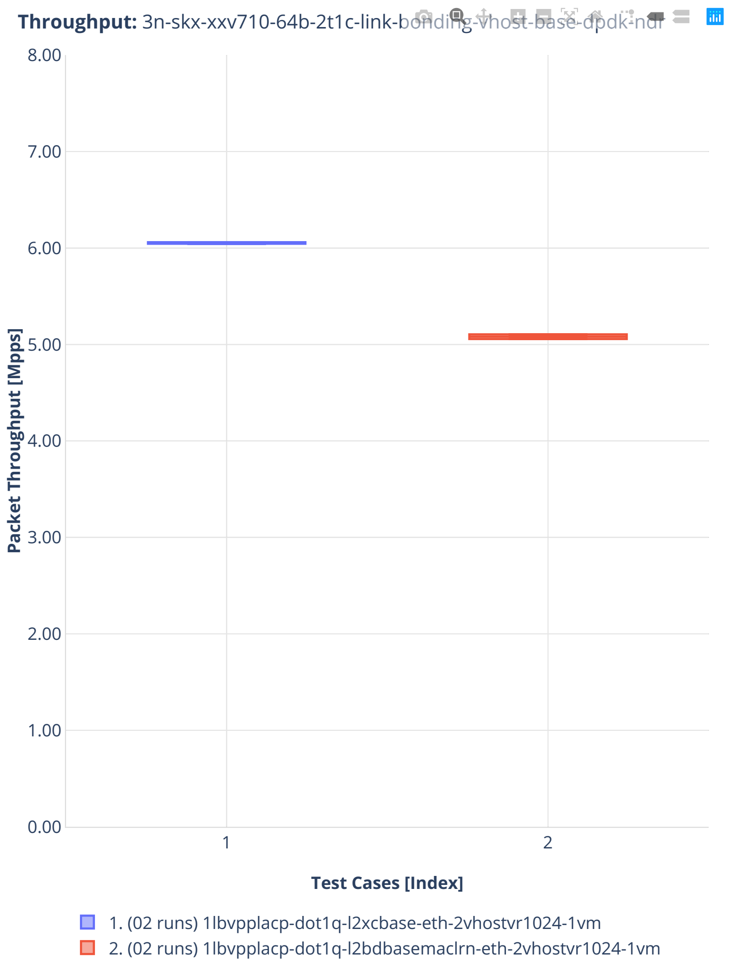
Throughput: 3n-skx-xxv710-64b-2t1c-vhost-base-dpdk-pdr
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Packet Throughput [Mpps]
1
2
3
4
5
6
Test Cases [Index]
1. (02 runs) dot1q-l2xcbase-eth-2vhostvr1024-1vm
2. (02 runs) eth-l2xcbase-eth-2vhostvr1024-1vm
3. (02 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm
4. (02 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm
5. (02 runs) ethip4-ip4base-eth-2vhostvr1024-1vm
6. (02 runs) ethip4vxlan-l2bdbasemaclrn-eth-2vhostvr1024-1vm

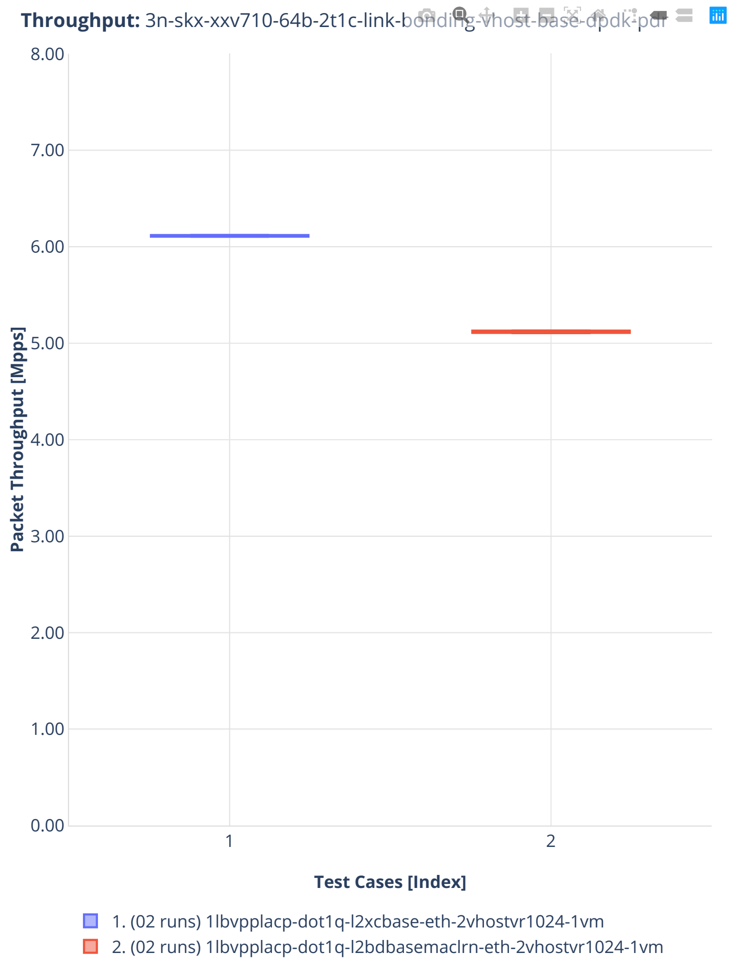
64b-2t1c-vhost-base-dpdk-vpp



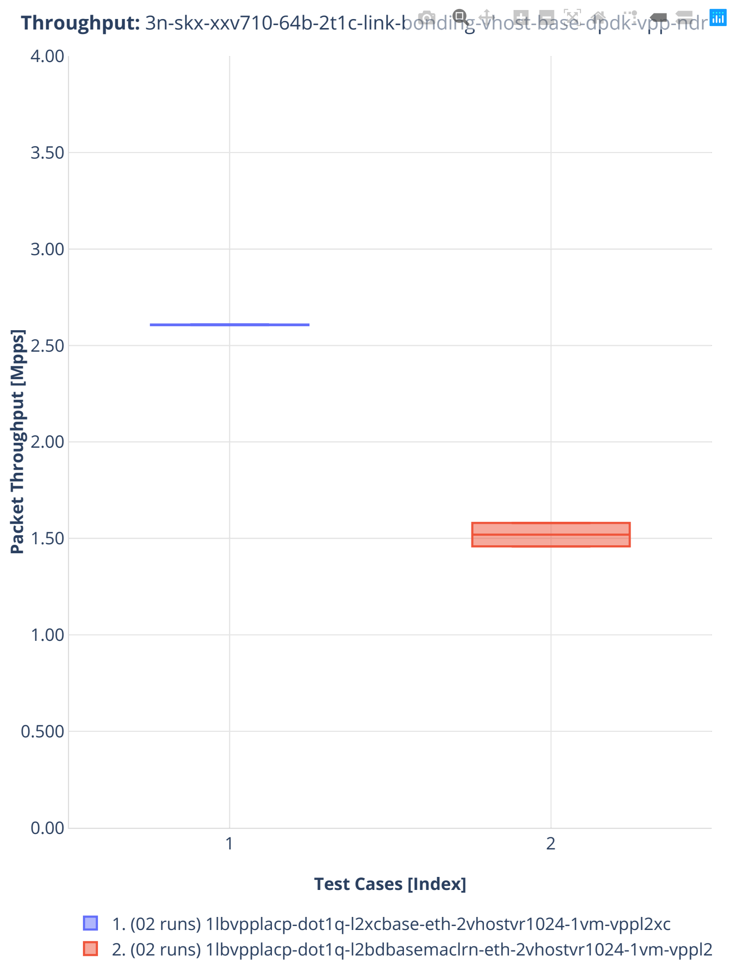

Throughput: 3n-skx-xxv710-64b-2t1c-vhost-base-dpdk-vpp-pdr
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Packet Throughput [Mpps]
1
2
3
4
5
Test Cases [Index]
1. (02 runs) dot1q-l2xcbase-eth-2vhostvr1024-1vm-vppl2xc
2. (02 runs) eth-l2xcbase-eth-2vhostvr1024-1vm-vppl2xc
3. (02 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
4. (02 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
5. (02 runs) ethip4-ip4base-eth-2vhostvr1024-1vm-vppip4

64b-2t1c-link-bonding-vhost-base-dpdk-testpmd



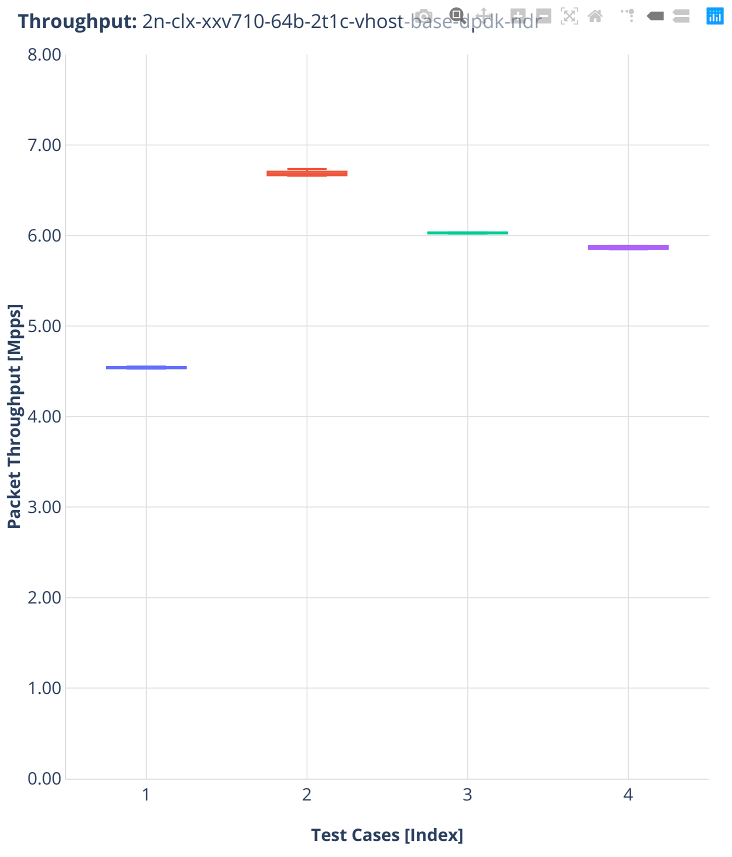

Throughput: 3n-skx-xxv710-64b-2t1c-link-bonding-vhost-base-dpdk-pdr
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Packet Throughput [Mpps]
1
2
Test Cases [Index]
1. (02 runs) 1lbvpplacp-dot1q-l2xcbase-eth-2vhostvr1024-1vm
2. (02 runs) 1lbvpplacp-dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm

64b-2t1c-link-bonding-vhost-base-dpdk-vpp

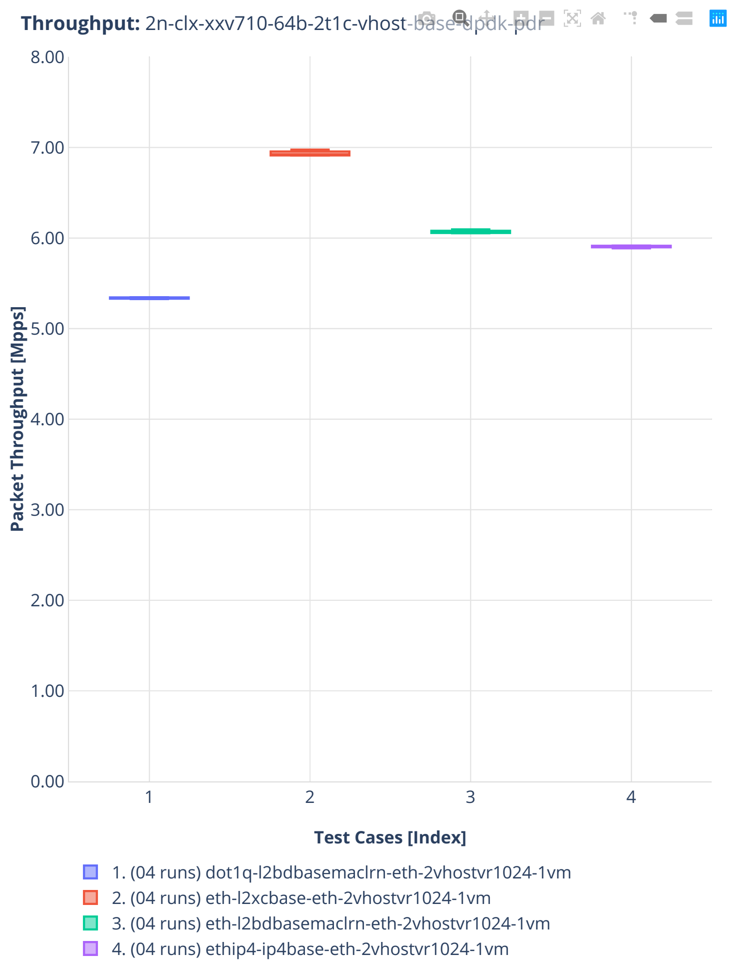


2n-clx-xxv710

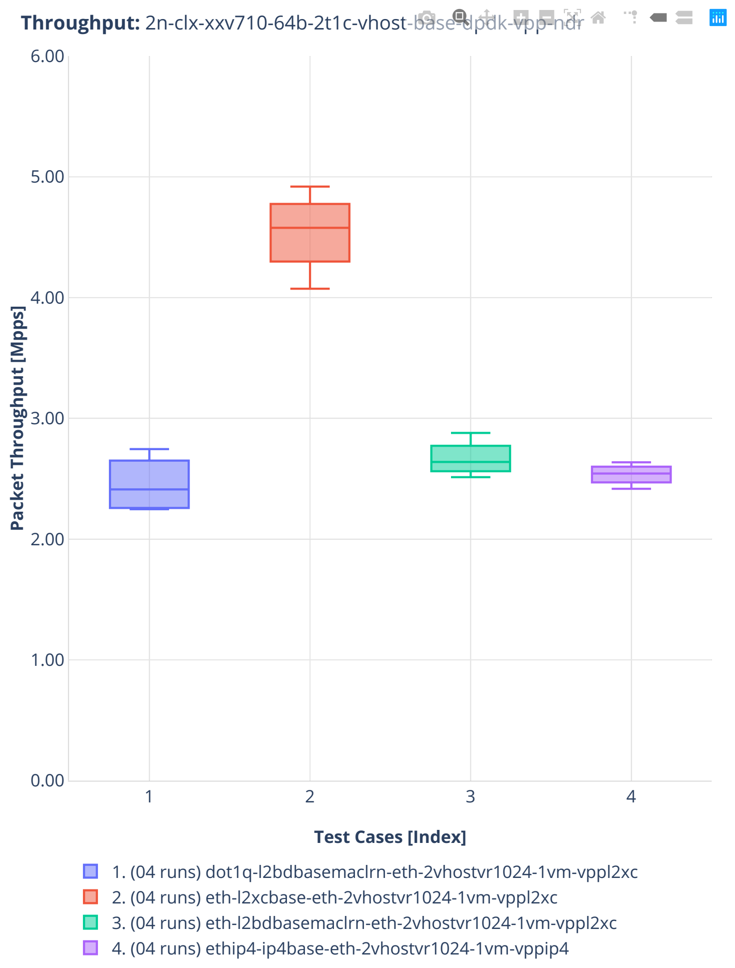
64b-2t1c-vhost-base-dpdk-testpmd

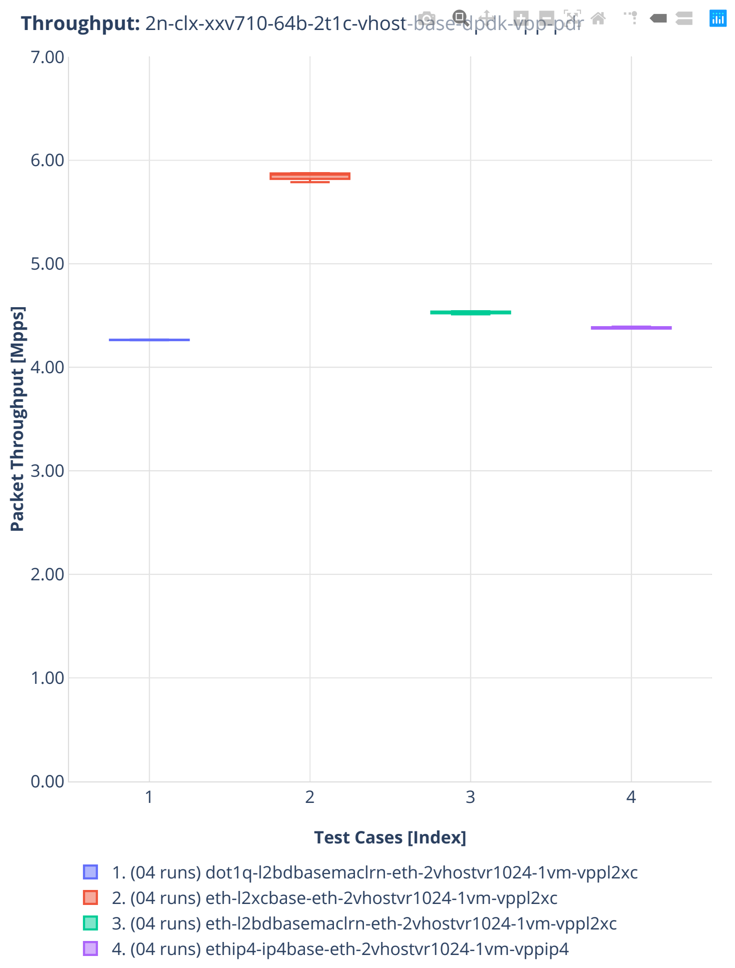


🟦 1. (04 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm
🟥 2. (04 runs) eth-l2xcbase-eth-2vhostvr1024-1vm
🟩 3. (04 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm
🟪 4. (04 runs) ethip4-ip4base-eth-2vhostvr1024-1vm


Throughput: 2n-clx-xxv710-64b-2t1c-vhost-base-dpdk-pdr
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Packet Throughput [Mpps]
1
2
3
4
Test Cases [Index]
1. (04 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm
2. (04 runs) eth-l2xcbase-eth-2vhostvr1024-1vm
3. (04 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm
4. (04 runs) ethip4-ip4base-eth-2vhostvr1024-1vm

64b-2t1c-vhost-base-dpdk-vpp




Throughput: 2n-clx-xxv710-64b-2t1c-vhost-base-dpdk-vpp-pdr
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Packet Throughput [Mpps]
1
2
3
4
Test Cases [Index]
1. (04 runs) dot1q-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
2. (04 runs) eth-l2xcbase-eth-2vhostvr1024-1vm-vppl2xc
3. (04 runs) eth-l2bdbasemaclrn-eth-2vhostvr1024-1vm-vppl2xc
4. (04 runs) ethip4-ip4base-eth-2vhostvr1024-1vm-vppip4

2.3.5 LXC/DRC Container Memif

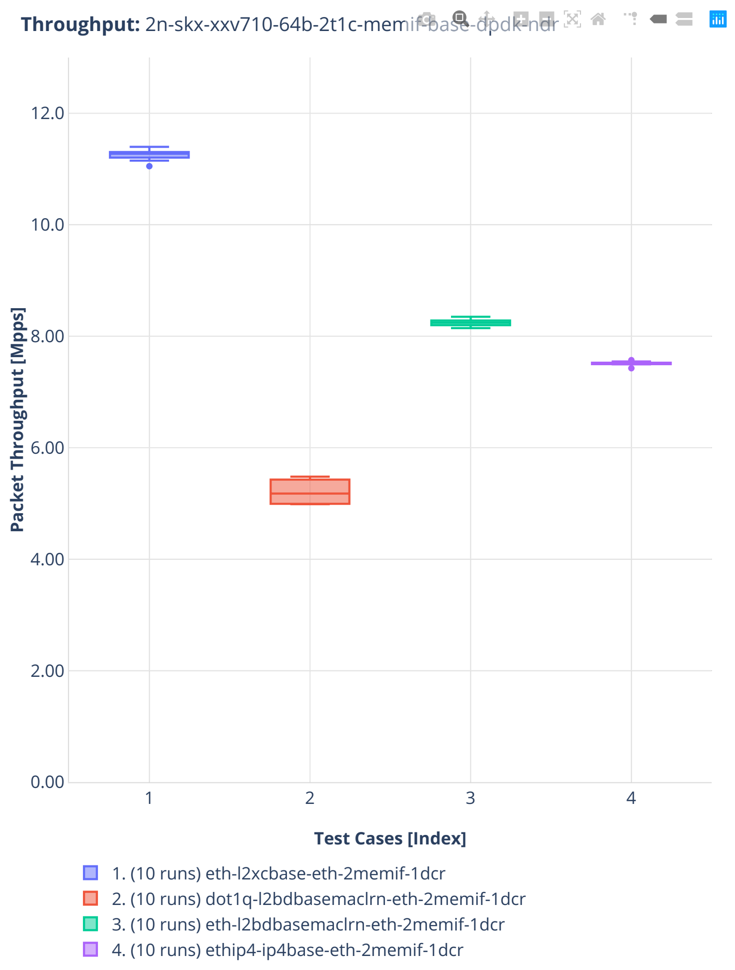
Following sections include summary graphs of VPP Phy-to-Phy performance with Container memif Connections, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

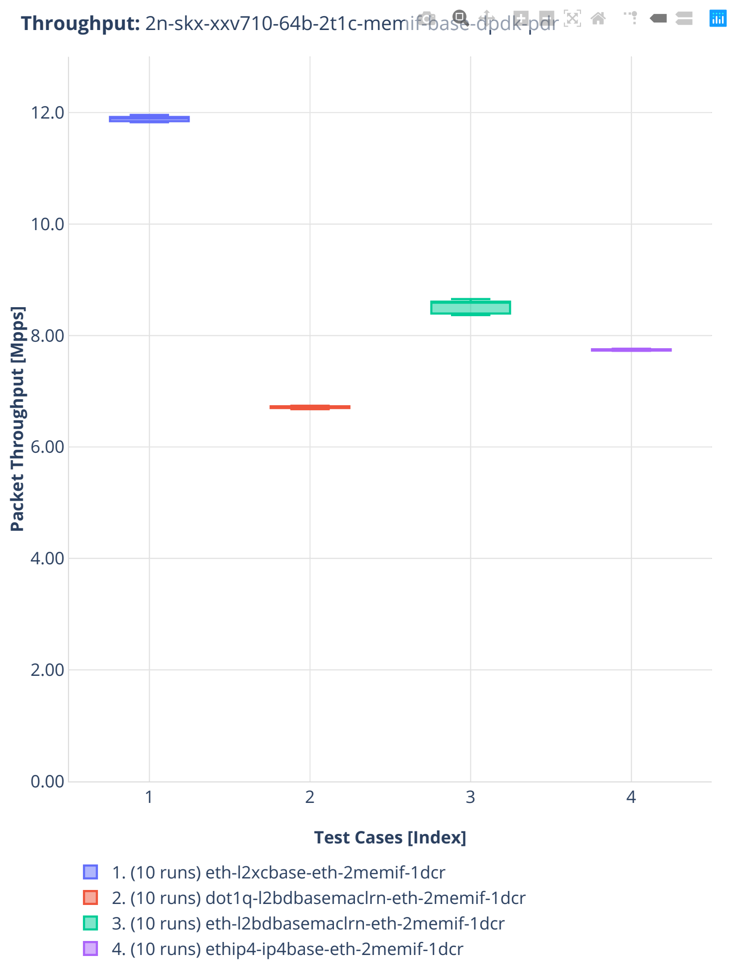
CSIT source code for the test cases used for plots can be found in CSIT git repository[45].

[45] https://git.fd.io/csit/tree/tests/vpp/perf/container_memif?h=rls1908_2

2n-skx-xxv710

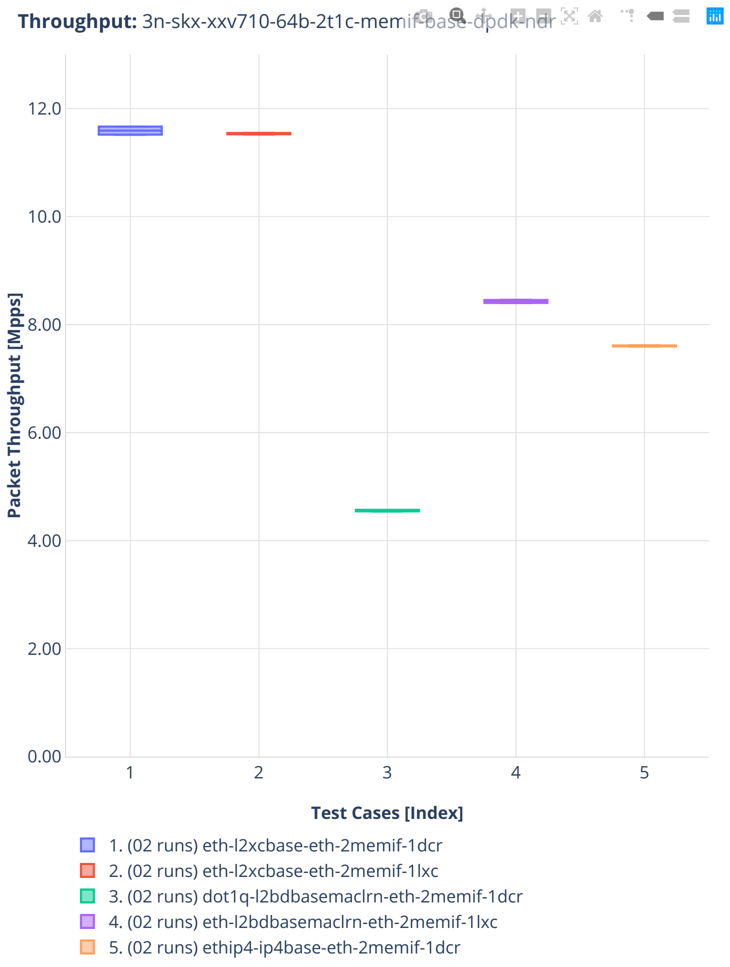
64b-2t1c-memif-base-dpdk

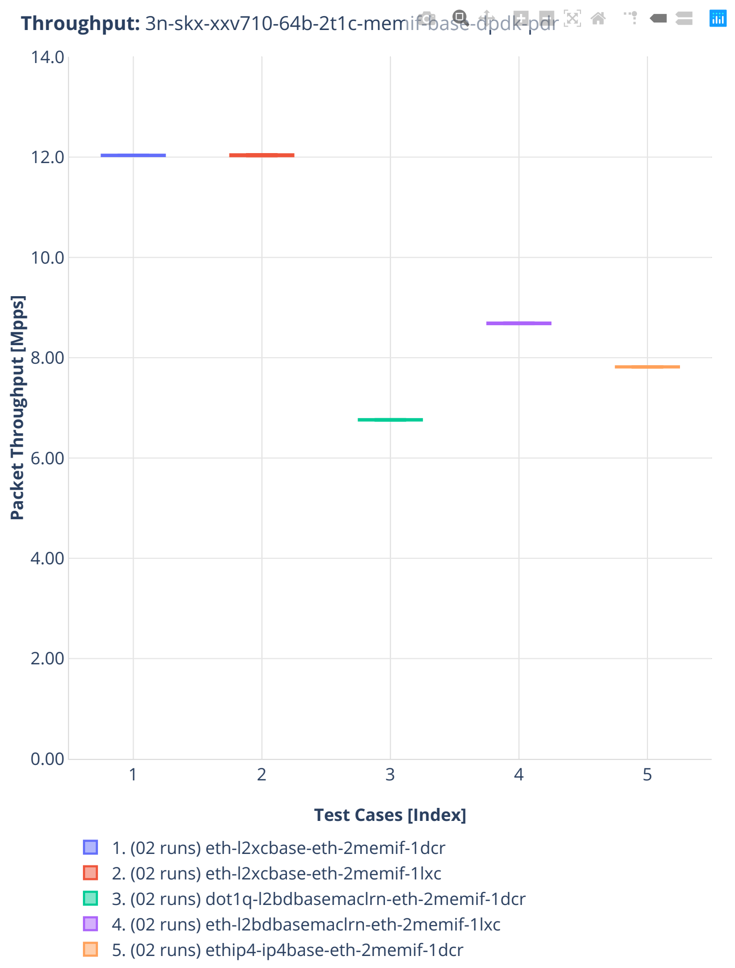



Throughput: 2n-skx-xxv710-64b-2t1c-memif-base-dpdk-pdr
Packet Throughput [Mpps]
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
Test Cases [Index]
1. (10 runs) eth-l2xcbase-eth-2memif-1dcr
2. (10 runs) dot1q-l2bdbasemaclrn-eth-2memif-1dcr
3. (10 runs) eth-l2bdbasemaclrn-eth-2memif-1dcr
4. (10 runs) ethip4-ip4base-eth-2memif-1dcr

3n-skx-xxv710

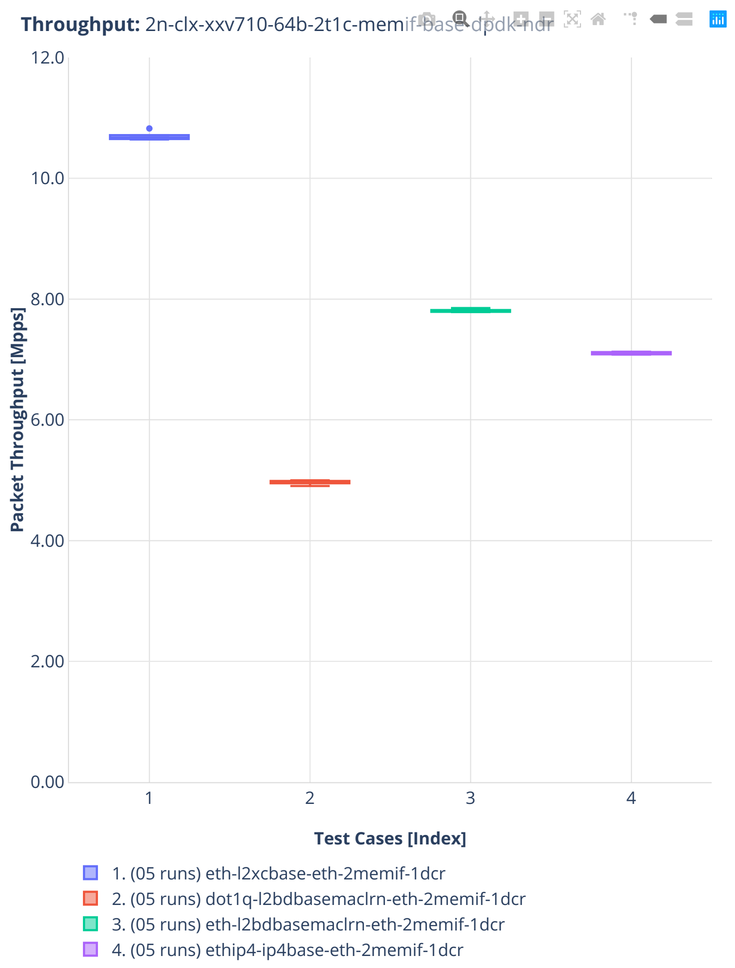
64b-2t1c-memif-base-dpdk

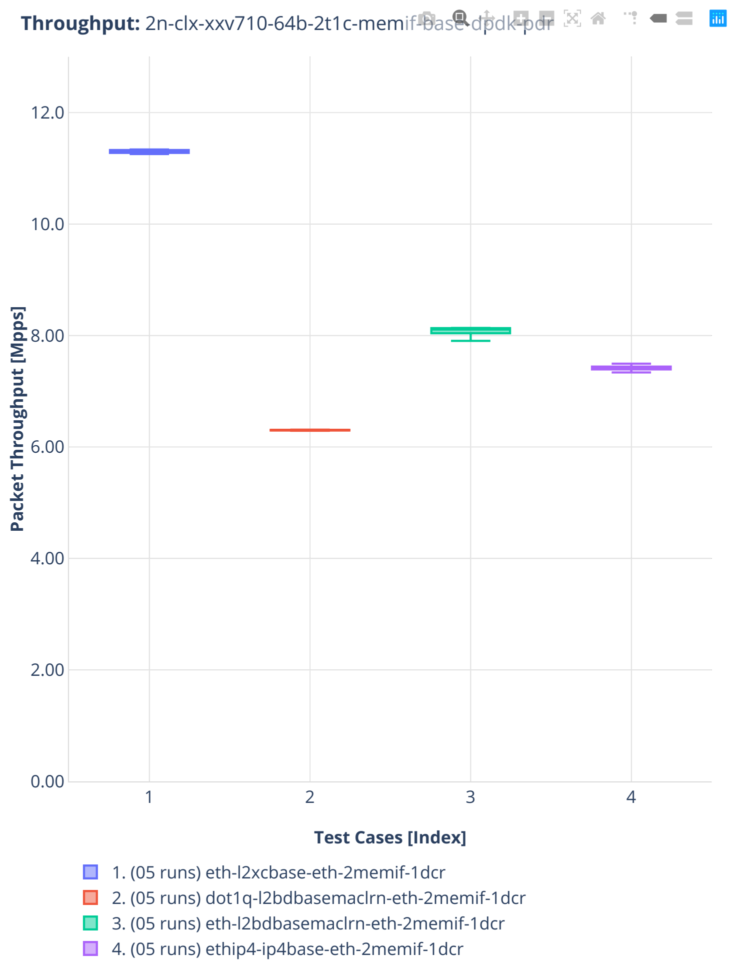



Throughput: 3n-skx-xxv710-64b-2t1c-memif-base-dpdk-pdr
Packet Throughput [Mpps]
14.0
12.0
10.0
8.00
6.00
4.00
2.00
0.00
1
2
3
4
5
Test Cases [Index]
1. (02 runs) eth-l2xcbase-eth-2memif-1dcr
2. (02 runs) eth-l2xcbase-eth-2memif-1lxc
3. (02 runs) dot1q-l2bdbasemaclrn-eth-2memif-1dcr
4. (02 runs) eth-l2bdbasemaclrn-eth-2memif-1lxc
5. (02 runs) ethip4-ip4base-eth-2memif-1dcr

2n-clx-xxv710

64b-2t1c-memif-base-dpdk




Throughput: 2n-clx-xxv710-64b-2t1c-memif-base-dpdk-pdr
12.0
10.0
8.00
6.00
4.00
2.00
0.00
Packet Throughput [Mpps]
1
2
3
4
Test Cases [Index]
1. (05 runs) eth-l2xcbase-eth-2memif-1dcr
2. (05 runs) dot1q-l2bdbasemaclrn-eth-2memif-1dcr
3. (05 runs) eth-l2bdbasemaclrn-eth-2memif-1dcr
4. (05 runs) ethip4-ip4base-eth-2memif-1dcr

2.3.6 IPSec IPv4 Routing

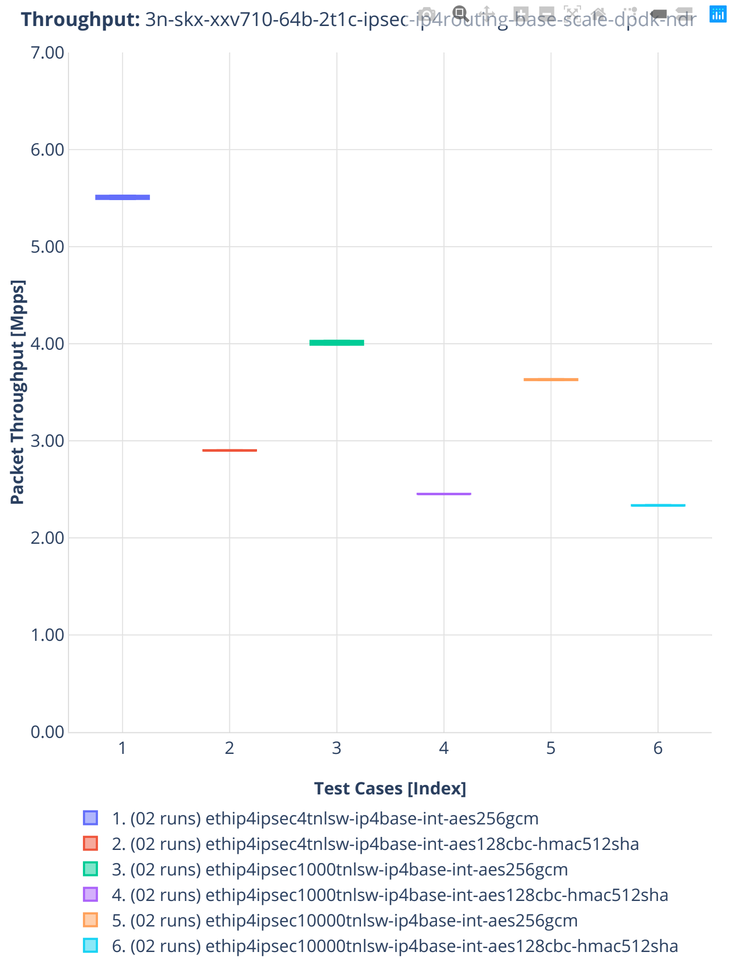
Following sections include summary graphs of VPP Phy-to-Phy performance with IPSec encryption used in combination with IPv4 routed-forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). VPP IPSec encryption is accelerated using DPDK cryptodev library driving Intel Quick Assist (QAT) crypto PCIe hardware cards. Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

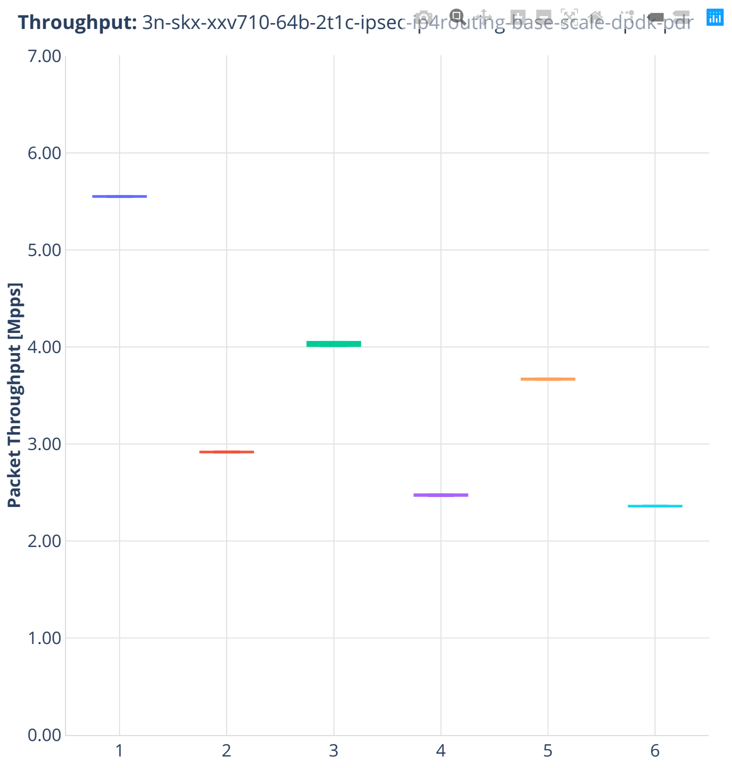
CSIT source code for the test cases used for plots can be found in CSIT git repository[46].

[46] https://git.fd.io/csit/tree/tests/vpp/perf/crypto?h=rls1908_2

3n-skx-xxv710

64b-2t1c-ipsec-ip4routing-base-scale-dpdk





🟦 1. (02 runs) ethip4ipsec4tnlsw-ip4base-int-aes256gcm
🟥 2. (02 runs) ethip4ipsec4tnlsw-ip4base-int-aes128cbc-hmac512sha
🟩 3. (02 runs) ethip4ipsec1000tnlsw-ip4base-int-aes256gcm
🟪 4. (02 runs) ethip4ipsec1000tnlsw-ip4base-int-aes128cbc-hmac512sha
🟧 5. (02 runs) ethip4ipsec10000tnlsw-ip4base-int-aes256gcm
🟦 6. (02 runs) ethip4ipsec10000tnlsw-ip4base-int-aes128cbc-hmac512sha

2.4 Comparisons

2.4.1 Current vs. Previous Release

Relative comparison of VPP packet throughput (NDR, PDR and MRR) between VPP-19.08.2 release and VPP-19.08.1 release (measured for CSIT-1908.2 and CSIT-1908.1 respectively) is calculated from results of tests running on 2-node Intel Xeon Skylake (2n-skx), 3-node Intel Xeon Skylake (3n-skx), 3-Node Intel Xeon Haswell (3n-hsw), 2-node Intel Atom Denverton (2n-dnv), 3-node Intel Atom Denverton (3n-dnv), 3-node Arm TaiShan (3n-tsh) testbeds, in 1-core, 2-core and 4-core (MRR only) configurations.

Listed mean and standard deviation values are computed based on a series of the same tests executed against respective VPP releases to verify test results repeatability, with percentage change calculated for mean values. Note that the standard deviation is quite high for a small number of packet throughput tests, what indicates poor test results repeatability and makes the relative change of mean throughput value not fully representative for these tests. The root causes behind poor results repeatability vary between the test cases.

Note: Test results have been generated by

- FD.io test executor vpp performance job 2n-skx[47],
- FD.io test executor vpp performance job 3n-skx[48],
- FD.io test executor vpp performance job 2n-clx[49]

with RF result files csit-vpp-perf-1908_2-*.zip archived here.

2n-skx

NDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 2t1c NDR comparison
- ASCII 2t1c NDR comparison
- CSV 2t1c NDR comparison

PDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 2t1c PDR comparison
- ASCII 2t1c PDR comparison
- CSV 2t1c PDR comparison

3n-skx

NDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

[47] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-skx
[48] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-3n-skx
[49] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-clx

- HTML 2t1c NDR comparison
- ASCII 2t1c NDR comparison
- CSV 2t1c NDR comparison

PDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 2t1c PDR comparison
- ASCII 2t1c PDR comparison
- CSV 2t1c PDR comparison

2n-clx

NDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 2t1c NDR comparison
- ASCII 2t1c NDR comparison
- CSV 2t1c NDR comparison

PDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 2t1c PDR comparison
- ASCII 2t1c PDR comparison
- CSV 2t1c PDR comparison

2.4.2 2n-Skx vs. 2n-Clx Testbeds

Relative comparison of VPP-19.08.2 release packet throughput (NDR, PDR and MRR) is calculated for the same tests executed on 2-Node Skylake (2n- skx) and 2-Node Cascade Lake (2n-clx) physical testbed types, in 1-core, 2-core and 4-core configurations.

Note: Test results have been generated by FD.io test executor vpp performance job 2n-skx[50] and FD.io test executor vpp performance job 2n-clx[51] with RF result files csit-vpp-perf-1908_2-*.zip archived here.

NDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 1c NDR comparison
- ASCII 1c NDR comparison
- CSV 1c NDR comparison

[50] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-skx
[51] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-clx

PDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 1c PDR comparison
- ASCII 1c PDR comparison
- CSV 1c PDR comparison

2.4.3 3n-Skx vs. 2n-Skx Testbeds

Relative comparison of VPP-19.08.2 release packet throughput (NDR, PDR and MRR) is calculated for the same tests executed on 3-Node Skylake (3n- skx) and 2-Node Skylake (2n-skx) physical testbed types, in 1-core, 2-core and 4-core configurations.

Note: Test results have been generated by FD.io test executor vpp performance job 3n-skx[52] and FD.io test executor vpp performance job 2n-skx[53] with RF result files csit-vpp-perf-1908_2-*.zip archived here.

NDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 1c NDR comparison
- ASCII 1c NDR comparison
- CSV 1c NDR comparison

PDR Comparison

Comparison tables in HTML, ASCII and CSV formats:

- HTML 1c PDR comparison
- ASCII 1c PDR comparison
- CSV 1c PDR comparison

2.5 Throughput Trending

In addition to reporting throughput comparison between VPP releases, CSIT provides continuous performance trending for VPP master branch:

1. Performance Dashboard[54]: per VPP test case throughput trend, trend compliance and summary of detected anomalies.
2. Trending Methodology[55]: throughput test metrics, trend calculations and anomaly classification (progression, regression).
3. VPP Trendline Graphs[56]: per VPP build MRR throughput measurements against the trendline with anomaly highlights and associated CSIT test jobs.

[52] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-3n-skx
[53] https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1908_2-2n-skx
[54] https://docs.fd.io/csit/master/trending/introduction/index.html
[55] https://docs.fd.io/csit/master/trending/methodology/index.html
[56] https://docs.fd.io/csit/master/trending/trending/index.html

2.6 Test Environment

2.6.1 Physical Testbeds

FD.io CSIT performance tests are executed in physical testbeds hosted by LF for FD.io project. Two physical testbed topology types are used:

- **3-Node Topology**: Consisting of two servers acting as SUTs (Systems Under Test) and one server as TG (Traffic Generator), all connected in ring topology.
- **2-Node Topology**: Consisting of one server acting as SUTs and one server as TG both connected in ring topology.

Tested SUT servers are based on a range of processors including Intel Xeon Haswell-SP, Intel Xeon Skylake-SP, Intel Xeon Cascade Lake-SP, Arm, Intel Atom. More detailed description is provided in *Physical Testbeds* (page 3). Tested logical topologies are described in *Logical Topologies* (page 37).

2.6.2 Server Specifications

Complete technical specifications of compute servers used in CSIT physical testbeds are maintained in FD.io CSIT repository: FD.io CSIT testbeds - Xeon Cascade Lake[57], FD.io CSIT testbeds - Xeon Skylake, Arm, Atom[58] and FD.io CSIT Testbeds - Xeon Haswell[59].

2.6.3 Pre-Test Server Calibration

Number of SUT server sub-system runtime parameters have been identified as impacting data plane performance tests. Calibrating those parameters is part of FD.io CSIT pre-test activities, and includes measuring and reporting following:

1. System level core jitter - measure duration of core interrupts by Linux in clock cycles and how often interrupts happen. Using CPU core jitter tool[60].
2. Memory bandwidth - measure bandwidth with Intel MLC tool[61].
3. Memory latency - measure memory latency with Intel MLC tool.
4. Cache latency at all levels (L1, L2, and Last Level Cache) - measure cache latency with Intel MLC tool.

Measured values of listed parameters are especially important for repeatable zero packet loss throughput measurements across multiple system instances. Generally they come useful as a background data for comparing data plane performance results across disparate servers.

Following sections include measured calibration data for testbeds.

2.6.4 Calibration Data - Skylake

Following sections include sample calibration data measured on s11-t31-sut1 server running in one of the Intel Xeon Skylake testbeds as specified in FD.io CSIT testbeds - Xeon Skylake, Arm, Atom[62].

Calibration data obtained from all other servers in Skylake testbeds shows the same or similar values.

[57] https://git.fd.io/csit/tree/docs/lab/testbeds_sm_clx_hw_bios_cfg.md?h=rls1908_2
[58] https://git.fd.io/csit/tree/docs/lab/testbeds_sm_skx_hw_bios_cfg.md?h=rls1908_2
[59] https://git.fd.io/csit/tree/docs/lab/testbeds_ucs_hsw_hw_bios_cfg.md?h=rls1908_2
[60] https://git.fd.io/pma_tools/tree/jitter
[61] https://software.intel.com/en-us/articles/intelr-memory-latency-checker
[62] https://git.fd.io/csit/tree/docs/lab/testbeds_sm_skx_hw_bios_cfg.md?h=rls1908_2

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-72-generic root=UUID=e05120bb-7127-43db-b1e3-a66edd4c43bd ro␣
↪isolcpus=1-27,29-55,57-83,85-111 nohz_full=1-27,29-55,57-83,85-111 rcu_nocbs=1-27,29-55,57-83,85-
↪111 numa_balancing=disable intel_pstate=disable intel_iommu=on iommu=pt nmi_watchdog=0 audit=0␣
↪nosoftlockup processor.max_cstate=1 intel_idle.max_cstate=1 hpet=disable tsc=reliable mce=off␣
↪console=tty0 console=ttyS0,115200n8
```

Linux uname

```
$ uname -a
Linux s3-t21-sut1 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64 x86_64 x86_
↪64 GNU/Linux
```

System-level Core Jitter

```
$ sudo taskset -c 3 /home/testuser/pma_tools/jitter/jitter -i 20
Linux Jitter testing program version 1.8
Iterations=20
The pragram will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Timings are in CPU Core cycles
Inst_Min:    Minimum Excution time during the display update interval(default is ~1 second)
Inst_Max:    Maximum Excution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Excution time during rhe display update interval. This is the value of␣
↪interest
last_Exec:   The Excution time of last iteration just before the display update
Abs_Min:     Absolute Minimum Excution time since the program started or statistics were reset
Abs_Max:     Absolute Maximum Excution time since the program started or statistics were reset
tmp:         Cumulative value calcualted by the dummy function
Interval:    Time interval between the display updates in Core Cycles
Sample No:   Sample number

   Inst_Min   Inst_Max   Inst_jitter last_Exec  Abs_min    Abs_max      tmp       Interval    ␣
↪Sample No
    160022     171330      11308     160022     160022     171330    2538733568 3204142750         ␣
↪1
    160022     167294       7272     160026     160022     171330     328335360 3203873548         ␣
↪2
    160022     167560       7538     160026     160022     171330    2412904448 3203878736         ␣
↪3
    160022     169000       8978     160024     160022     171330     202506240 3203864588         ␣
↪4
    160022     166572       6550     160026     160022     171330    2287075328 3203866224         ␣
↪5
    160022     167460       7438     160026     160022     171330      76677120 3203854632         ␣
↪6
    160022     168134       8112     160024     160022     171330    2161246208 3203874674         ␣
↪7
    160022     169094       9072     160022     160022     171330    4245815296 3203878798         ␣
↪8
    160022     172460      12438     160024     160022     172460    2035417088 3204112010         ␣
↪9
    160022     167862       7840     160030     160022     172460    4119986176 3203856800        ␣
↪10
    160022     168398       8376     160024     160022     172460    1909587968 3203854192        ␣
↪11
```

(continues on next page)

(continued from previous page)

```
    160022     167548       7526     160024     160022     172460     3994157056 3203847442          ␣
↪12
    160022     167562       7540     160026     160022     172460     1783758848 3203862936          ␣
↪13
    160022     167604       7582     160024     160022     172460     3868327936 3203859346          ␣
↪14
    160022     168262       8240     160024     160022     172460     1657929728 3203851120          ␣
↪15
    160022     169700       9678     160024     160022     172460     3742498816 3203877690          ␣
↪16
    160022     170476      10454     160026     160022     172460     1532100608 3204088480          ␣
↪17
    160022     167798       7776     160024     160022     172460     3616669696 3203862072          ␣
↪18
    160022     166540       6518     160024     160022     172460     1406271488 3203836904          ␣
↪19
    160022     167516       7494     160024     160022     172460     3490840576 3203848120          ␣
↪20
```

Memory Bandwidth

```
$ sudo /home/testuser/mlc --bandwidth_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --bandwidth_matrix

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes
Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
                Numa node
Numa node        0       1
    0     107947.7    50951.5
    1      50834.6   108183.4
```

```
$ sudo /home/testuser/mlc --peak_injection_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --peak_injection_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Peak Injection Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :  215733.9
3:1 Reads-Writes :  182141.9
2:1 Reads-Writes :  178615.7
1:1 Reads-Writes :  149911.3
Stream-triad like:  159533.6
```

```
$ sudo /home/testuser/mlc --max_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --max_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Maximum Memory Bandwidths for the system
```

(continues on next page)

(continued from previous page)

```
Will take several minutes to complete as multiple injection rates will be tried to get the best␣
↪bandwidth
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :  216875.73
3:1 Reads-Writes :  182615.14
2:1 Reads-Writes :  178745.67
1:1 Reads-Writes :  149485.27
Stream-triad like:  180057.87
```

Memory Latency

```
$ sudo /home/testuser/mlc --latency_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --latency_matrix

Using buffer size of 2000.000MB
Measuring idle latencies (in ns)...
             Numa node
Numa node    0       1
    0      81.4    131.1
    1     131.1     81.3
```

```
$ sudo /home/testuser/mlc --idle_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --idle_latency

Using buffer size of 2000.000MB
Each iteration took 202.0 core clocks ( 80.8    ns)
```

```
$ sudo /home/testuser/mlc --loaded_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --loaded_latency

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Loaded Latencies for the system
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  282.66   215712.8
 00002  282.14   215757.4
 00008  280.21   215868.1
 00015  279.20   216313.2
 00050  275.25   216643.0
 00100  227.05   215075.0
 00200  121.92   160242.9
 00300  101.21   111587.4
 00400   95.48    85019.7
 00500   94.46    68717.3
 00700   92.27    49742.2
 01000   91.03    35264.8
 01300   90.11    27396.3
 01700   89.34    21178.7
 02500   90.15    14672.8
```

(continues on next page)

(continued from previous page)

```
03500   89.00    10715.7
05000   82.00     7788.2
09000   81.46     4684.0
20000   81.40     2541.9
```

L1/L2/LLC Latency

```
$ sudo /home/testuser/mlc --c2c_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --c2c_latency

Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency    53.7
Local Socket L2->L2 HITM latency    53.7
Remote Socket L2->L2 HITM latency (data address homed in writer socket)
                     Reader Numa Node
Writer Numa Node        0       1
            0           -   113.9
            1       113.9       -
Remote Socket L2->L2 HITM latency (data address homed in reader socket)
                     Reader Numa Node
Writer Numa Node        0       1
            0           -   177.9
            1       177.6       -
```

Spectre and Meltdown Checks

Following section displays the output of a running shell script to tell if system is vulnerable against the several “speculative execution” CVEs that were made public in 2018. Script is available on Spectre & Meltdown Checker Github[63].

```
Spectre and Meltdown mitigation detection tool v0.43

awk: cannot open bash (No such file or directory)
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
CPU is Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
 * Indirect Branch Restricted Speculation (IBRS)
   * SPEC_CTRL MSR is available: YES
   * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
 * Indirect Branch Prediction Barrier (IBPB)
   * PRED_CMD MSR is available: YES
   * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
 * Single Thread Indirect Branch Predictors (STIBP)
   * SPEC_CTRL MSR is available: YES
   * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
 * Speculative Store Bypass Disable (SSBD)
   * CPU indicates SSBD capability: YES (Intel SSBD)
 * L1 data cache invalidation
   * FLUSH_CMD MSR is available: YES
   * CPU indicates L1D flush capability: YES (L1D flush feature bit)
 * Microarchitectural Data Sampling
   * VERW instruction is available: YES (MD_CLEAR feature bit)
```

(continues on next page)

[63] https://github.com/speed47/spectre-meltdown-checker

(continued from previous page)

```
 * Enhanced IBRS (IBRS_ALL)
   * CPU indicates ARCH_CAPABILITIES MSR availability: NO
   * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: NO
 * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): NO
 * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
 * CPU/Hypervisor indicates L1D flushing is not necessary on this system: NO
 * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
 * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): NO
 * CPU explicitly indicates not being vulnerable to TSX Asynchronous Abort (TAA_NO): NO
 * CPU explicitly indicates not being vulnerable to iTLB Multihit (PSCHANGE_MSC_NO): NO
 * CPU explicitly indicates having MSR for TSX control (TSX_CTRL_MSR): NO
 * CPU supports Transactional Synchronization Extensions (TSX): YES (RTM feature bit)
 * CPU supports Software Guard Extensions (SGX): NO
 * CPU microcode is known to cause stability problems: NO (model 0x55 family 0x6 stepping 0x4 ucode␣
↪0x2000064 cpuid 0x50654)
 * CPU microcode is the latest known available version: awk: cannot open bash (No such file or␣
↪directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
 * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
 * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
 * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): YES
 * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
 * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
 * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
 * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): YES
 * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): YES
 * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪YES
 * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)):␣
↪YES
 * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): YES
 * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): YES
 * Vulnerable to CVE-2019-11135 (ZombieLoad V2, TSX Asynchronous Abort (TAA)): YES
 * Vulnerable to CVE-2018-12207 (No eXcuses, iTLB Multihit, machine check exception on page size␣
↪changes (MCEPSC)): YES

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: usercopy/swapgs barriers and __user␣
↪pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Full generic retpoline, IBPB:␣
↪conditional, IBRS_FW, STIBP: conditional, RSB filling)
* Mitigation 1
 * Kernel is compiled with IBRS support: YES
   * IBRS enabled and active: YES (for firmware code only)
 * Kernel is compiled with IBPB support: YES
   * IBPB enabled and active: YES
* Mitigation 2
 * Kernel has branch predictor hardening (arm): NO
 * Kernel compiled with retpoline option: YES
   * Kernel compiled with a retpoline-aware compiler: YES (kernel reports full retpoline␣
↪compilation)
 * Kernel supports RSB filling: YES
```

(continues on next page)

(continued from previous page)

```
> STATUS: NOT VULNERABLE (Full retpoline + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Mitigation: PTI)
* Kernel supports Page Table Isolation (PTI): YES
 * PTI enabled and active: YES
 * Reduced performance impact of PTI: YES (CPU supports INVPCID, performance impact of PTI will be␣
↪greatly reduced)
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (Mitigation: PTI)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: YES
> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: YES (Mitigation: Speculative Store Bypass disabled via␣
↪prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES (systemd-journald systemd-logind␣
↪systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)
> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
* Mitigated according to the /sys interface: YES (Mitigation: PTE Inversion; VMX: conditional cache␣
↪flushes, SMT vulnerable)
* Kernel supports PTE inversion: YES (found in kernel image)
* PTE inversion enabled and active: YES
> STATUS: NOT VULNERABLE (Mitigation: PTE Inversion; VMX: conditional cache flushes, SMT vulnerable)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* Information from the /sys interface: Mitigation: PTE Inversion; VMX: conditional cache flushes,␣
↪SMT vulnerable
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
 * EPT is disabled: NO
* Mitigation 2
 * L1D flush is supported by kernel: YES (found flush_l1d in /proc/cpuinfo)
 * L1D flush enabled: YES (conditional flushes)
 * Hardware-backed L1D flush supported: YES (performance impact of the mitigation will be greatly␣
↪reduced)
 * Hyper-Threading (SMT) is enabled: YES
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
```

(continues on next page)

(continued from previous page)

```
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2019-11135 aka ZombieLoad V2, TSX Asynchronous Abort (TAA)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
* TAA mitigation is supported by kernel: YES (found tsx_async_abort in kernel image)
* TAA mitigation enabled and active: YES (Mitigation: Clear CPU buffers; SMT vulnerable)
> STATUS: NOT VULNERABLE (Mitigation: Clear CPU buffers; SMT vulnerable)

CVE-2018-12207 aka No eXcuses, iTLB Multihit, machine check exception on page size changes (MCEPSC)
* Mitigated according to the /sys interface: YES (KVM: Mitigation: Split huge pages)
* This system is a host running a hypervisor: NO
* iTLB Multihit mitigation is supported by kernel: YES (found itlb_multihit in kernel image)
* iTLB Multihit mitigation enabled and active: YES (KVM: Mitigation: Split huge pages)
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:OK CVE-2018-3639:OK CVE-
↪2018-3615:OK CVE-2018-3620:OK CVE-2018-3646:OK CVE-2018-12126:OK CVE-2018-12130:OK CVE-2018-
↪12127:OK CVE-2019-11091:OK CVE-2019-11135:OK CVE-2018-12207:OK
```

2.6.5 Calibration Data - Cascade Lake

Following sections include sample calibration data measured on s32-t27-sut1 server running in one of the Intel Xeon Skylake testbeds as specified in FD.io CSIT testbeds - Xeon Cascade Lake[64].

Calibration data obtained from all other servers in Cascade Lake testbeds shows the same or similar values.

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-72-generic root=UUID=1d03969e-a2a0-41b2-a97e-1cc171b07e88 ro␣
↪isolcpus=1-23,25-47,49-71,73-95 nohz_full=1-23,25-47,49-71,73-95 rcu_nocbs=1-23,25-47,49-71,73-95␣
↪numa_balancing=disable intel_pstate=disable intel_iommu=on iommu=pt nmi_watchdog=0 audit=0␣
↪nosoftlockup processor.max_cstate=1 intel_idle.max_cstate=1 hpet=disable tsc=reliable mce=off␣
↪console=tty0 console=ttyS0,115200n8
```

[64] https://git.fd.io/csit/tree/docs/lab/testbeds_sm_clx_hw_bios_cfg.md?h=rls1908_2

Linux uname

```
$ uname -a
Linux s32-t27-sut1 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64 x86_64 x86_
↪64 GNU/Linux
```

System-level Core Jitter

```
$ sudo taskset -c 3 /home/testuser/pma_tools/jitter/jitter -i 30
Linux Jitter testing program version 1.9
Iterations=30
The pragram will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Thread affinity will be set to core_id:7
Timings are in CPU Core cycles
Inst_Min:    Minimum Excution time during the display update interval(default is ~1 second)
Inst_Max:    Maximum Excution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Excution time during rhe display update interval. This is the value of␣
↪interest
last_Exec:   The Excution time of last iteration just before the display update
Abs_Min:     Absolute Minimum Excution time since the program started or statistics were reset
Abs_Max:     Absolute Maximum Excution time since the program started or statistics were reset
tmp:         Cumulative value calcualted by the dummy function
Interval:    Time interval between the display updates in Core Cycles
Sample No:   Sample number

Inst_Min,Inst_Max,Inst_jitter,last_Exec,Abs_min,Abs_max,tmp,Interval,Sample No
160022,167590,7568,160026,160022,167590,2057568256,3203711852,1
160022,170628,10606,160024,160022,170628,4079222784,3204010824,2
160022,169824,9802,160024,160022,170628,1805910016,3203812064,3
160022,168832,8810,160030,160022,170628,3827564544,3203792594,4
160022,168248,8226,160026,160022,170628,1554251776,3203765920,5
160022,167834,7812,160028,160022,170628,3575906304,3203761114,6
160022,167442,7420,160024,160022,170628,1302593536,3203769250,7
160022,169120,9098,160028,160022,170628,3324248064,3203853340,8
160022,170710,10688,160024,160022,170710,1050935296,3203985878,9
160022,167952,7930,160024,160022,170710,3072589824,3203733756,10
160022,168314,8292,160030,160022,170710,799277056,3203741152,11
160022,169672,9650,160024,160022,170710,2820931584,3203739910,12
160022,168684,8662,160024,160022,170710,547618816,3203727336,13
160022,168246,8224,160024,160022,170710,2569273344,3203739052,14
160022,168134,8112,160030,160022,170710,295960576,3203735874,15
160022,170230,10208,160024,160022,170710,2317615104,3203996356,16
160022,167190,7168,160024,160022,170710,44302336,3203713628,17
160022,167304,7282,160024,160022,170710,2065956864,3203717954,18
160022,167500,7478,160024,160022,170710,4087611392,3203706674,19
160022,167302,7280,160024,160022,170710,1814298624,3203726452,20
160022,167266,7244,160024,160022,170710,3835953152,3203702804,21
160022,167820,7798,160022,160022,170710,1562640384,3203719138,22
160022,168100,8078,160024,160022,170710,3584294912,3203716636,23
160022,170408,10386,160024,160022,170710,1310982144,3203946958,24
160022,167276,7254,160024,160022,170710,3332636672,3203706236,25
160022,167052,7030,160024,160022,170710,1059323904,3203696444,26
160022,170322,10300,160024,160022,170710,3080978432,3203747514,27
160022,167332,7310,160024,160022,170710,807665664,3203716210,28
160022,167426,7404,160026,160022,170710,2829320192,3203700630,29
160022,168840,8818,160024,160022,170710,556007424,3203727658,30
```

Memory Bandwidth

```
$ sudo /home/testuser/mlc --bandwidth_matrix
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --bandwidth_matrix

Using buffer size of 100.000MiB/thread for reads and an additional 100.000MiB/thread for writes
Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
                Numa node
Numa node            0       1
       0        122097.7     51327.9
       1        51309.2      122005.5
```

```
$ sudo /home/testuser/mlc --peak_injection_bandwidth
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --peak_injection_bandwidth

Using buffer size of 100.000MiB/thread for reads and an additional 100.000MiB/thread for writes

Measuring Peak Injection Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :      243159.4
3:1 Reads-Writes :      219132.5
2:1 Reads-Writes :      216603.1
1:1 Reads-Writes :      203713.0
Stream-triad like:      193790.8
```

```
$ sudo /home/testuser/mlc --max_bandwidth
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --max_bandwidth

Using buffer size of 100.000MiB/thread for reads and an additional 100.000MiB/thread for writes

Measuring Maximum Memory Bandwidths for the system
Will take several minutes to complete as multiple injection rates will be tried to get the best␣
↪bandwidth
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :      244114.27
3:1 Reads-Writes :      219441.97
2:1 Reads-Writes :      216603.72
1:1 Reads-Writes :      203679.09
Stream-triad like:      214902.80
```

Memory Latency

```
$ sudo /home/testuser/mlc --latency_matrix
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --latency_matrix

Using buffer size of 2000.000MiB
Measuring idle latencies (in ns)...
                Numa node
```

(continues on next page)

(continued from previous page)

```
Numa node            0       1
       0          81.2   130.2
       1         130.2    81.1
```

```
$ sudo /home/testuser/mlc --idle_latency
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --idle_latency

Using buffer size of 2000.000MiB
Each iteration took 186.1 core clocks ( 80.9    ns)
```

```
$ sudo /home/testuser/mlc --loaded_latency
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --loaded_latency

Using buffer size of 100.000MiB/thread for reads and an additional 100.000MiB/thread for writes

Measuring Loaded Latencies for the system
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  233.86   243421.9
 00002  230.61   243544.1
 00008  232.56   243394.5
 00015  229.52   244076.6
 00050  225.82   244290.6
 00100  161.65   236744.8
 00200  100.63   133844.0
 00300   96.84    90548.2
 00400   95.71    68504.3
 00500   95.68    55139.0
 00700   88.77    39798.4
 01000   84.74    28200.1
 01300   83.08    21915.5
 01700   82.27    16969.3
 02500   81.66    11810.6
 03500   81.98     8662.9
 05000   81.48     6306.8
 09000   81.17     3857.8
 20000   80.19     2179.9
```

L1/L2/LLC Latency

```
$ sudo /home/testuser/mlc --c2c_latency
Intel(R) Memory Latency Checker - v3.7
Command line parameters: --c2c_latency

Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency        55.5
Local Socket L2->L2 HITM latency        55.6
Remote Socket L2->L2 HITM latency (data address homed in writer socket)
                        Reader Numa Node
Writer Numa Node     0       1
            0        -   115.6
            1    115.6       -
Remote Socket L2->L2 HITM latency (data address homed in reader socket)
```

(continues on next page)

(continued from previous page)

```
                        Reader Numa Node
Writer Numa Node     0         1
            0        -     178.2
            1    178.4         -
```

Spectre and Meltdown Checks

Following section displays the output of a running shell script to tell if system is vulnerable against the several speculative execution CVEs that were made public in 2018. Script is available on Spectre & Meltdown Checker Github[65].

```
Spectre and Meltdown mitigation detection tool v0.43

awk: fatal: cannot open file `bash for reading (No such file or directory)
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
CPU is Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: YES (Intel SSBD)
  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: YES
    * CPU indicates L1D flush capability: YES (L1D flush feature bit)
  * Microarchitectural Data Sampling
    * VERW instruction is available: YES (MD_CLEAR feature bit)
  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: YES
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: YES
  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): YES
  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: YES
  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): YES
  * CPU explicitly indicates not being vulnerable to TSX Asynchronous Abort (TAA_NO): NO
  * CPU explicitly indicates not being vulnerable to iTLB Multihit (PSCHANGE_MSC_NO): NO
  * CPU explicitly indicates having MSR for TSX control (TSX_CTRL_MSR): YES
    * TSX_CTRL MSR indicates TSX RTM is disabled: YES
    * TSX_CTRL MSR indicates TSX CPUID bit is cleared: YES
  * CPU supports Transactional Synchronization Extensions (TSX): NO
  * CPU supports Software Guard Extensions (SGX): NO
  * CPU microcode is known to cause stability problems: NO (model 0x55 family 0x6 stepping 0x7␣
↪ucode 0x500002c cpuid 0x50657)
  * CPU microcode is the latest known available version: awk: fatal: cannot open file `bash for␣
↪reading (No such file or directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
```

(continues on next page)

[65] https://github.com/speed47/spectre-meltdown-checker

(continued from previous page)

```
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): NO
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): YES
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): YES
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪NO
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling␣
↪(MFBDS)): NO
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): NO
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): NO
  * Vulnerable to CVE-2019-11135 (ZombieLoad V2, TSX Asynchronous Abort (TAA)): NO
  * Vulnerable to CVE-2018-12207 (No eXcuses, iTLB Multihit, machine check exception on page size␣
↪changes (MCEPSC)): YES

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: usercopy/swapgs barriers and __user␣
↪pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Enhanced IBRS, IBPB: conditional, RSB␣
↪filling)
* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES (Enhanced flavor, performance impact will be greatly reduced)
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES
* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
  * Kernel supports RSB filling: YES
> STATUS: NOT VULNERABLE (Enhanced IBRS + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: UNKNOWN (dmesg truncated, please reboot and relaunch this script)
  * Reduced performance impact of PTI: YES (CPU supports INVPCID, performance impact of PTI will be␣
↪greatly reduced)
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: YES
> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: YES (Mitigation: Speculative Store Bypass disabled via␣
↪prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES (systemd-journald systemd-logind␣
↪systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)
```

(continues on next page)

(continued from previous page)

```
> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports PTE inversion: YES (found in kernel image)
* PTE inversion enabled and active: NO
> STATUS: NOT VULNERABLE (Not affected)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* Information from the /sys interface: Not affected
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: YES (found flush_l1d in /proc/cpuinfo)
  * L1D flush enabled: NO
  * Hardware-backed L1D flush supported: YES (performance impact of the mitigation will be greatly␣
↪reduced)
  * Hyper-Threading (SMT) is enabled: YES
> STATUS: NOT VULNERABLE (your kernel reported your CPU model as not vulnerable)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2019-11135 aka ZombieLoad V2, TSX Asynchronous Abort (TAA)
* Mitigated according to the /sys interface: YES (Mitigation: TSX disabled)
* TAA mitigation is supported by kernel: YES (found tsx_async_abort in kernel image)
* TAA mitigation enabled and active: YES (Mitigation: TSX disabled)
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12207 aka No eXcuses, iTLB Multihit, machine check exception on page size changes (MCEPSC)
* Mitigated according to the /sys interface: YES (KVM: Mitigation: Split huge pages)
```

(continues on next page)

(continued from previous page)

```
* This system is a host running a hypervisor: NO
* iTLB Multihit mitigation is supported by kernel: YES (found itlb_multihit in kernel image)
* iTLB Multihit mitigation enabled and active: YES (KVM: Mitigation: Split huge pages)
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:OK CVE-2018-3639:OK CVE-
↪2018-3615:OK CVE-2018-3620:OK CVE-2018-3646:OK CVE-2018-12126:OK CVE-2018-12130:OK CVE-2018-
↪12127:OK CVE-2019-11091:OK CVE-2019-11135:OK CVE-2018-12207:OK
```

```
awk: fatal: cannot open file `bash for reading (No such file or directory)
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
CPU is Intel(R) Xeon(R) Gold 6252N CPU @ 2.30GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: YES (Intel SSBD)
  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: YES
    * CPU indicates L1D flush capability: YES (L1D flush feature bit)
  * Microarchitectural Data Sampling
    * VERW instruction is available: YES (MD_CLEAR feature bit)
  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: YES
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: YES
  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): YES
  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: YES
  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): YES
  * CPU explicitly indicates not being vulnerable to TSX Asynchronous Abort (TAA_NO): NO
  * CPU explicitly indicates not being vulnerable to iTLB Multihit (PSCHANGE_MSC_NO): NO
  * CPU explicitly indicates having MSR for TSX control (TSX_CTRL_MSR): YES
    * TSX_CTRL MSR indicates TSX RTM is disabled: YES
    * TSX_CTRL MSR indicates TSX CPUID bit is cleared: YES
  * CPU supports Transactional Synchronization Extensions (TSX): NO
  * CPU supports Software Guard Extensions (SGX): NO
  * CPU microcode is known to cause stability problems: NO (family 0x6 model 0x55 stepping 0x7␣
↪ucode 0x500002c cpuid 0x50657)
  * CPU microcode is the latest known available version: awk: fatal: cannot open file `bash for␣
↪reading (No such file or directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): NO
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): YES
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): YES
```

(continues on next page)

(continued from previous page)

```
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪NO
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling␣
↪(MFBDS)): NO
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): NO
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): NO
  * Vulnerable to CVE-2019-11135 (ZombieLoad V2, TSX Asynchronous Abort (TAA)): NO
  * Vulnerable to CVE-2018-12207 (No eXcuses, iTLB Multihit, machine check exception on page size␣
↪changes (MCEPSC)): YES

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: usercopy/swapgs barriers and __user␣
↪pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Enhanced IBRS, IBPB: conditional, RSB␣
↪filling)
* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES (Enhanced flavor, performance impact will be greatly reduced)
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES
* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
  * Kernel supports RSB filling: YES
> STATUS: NOT VULNERABLE (Enhanced IBRS + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: UNKNOWN (dmesg truncated, please reboot and relaunch this script)
  * Reduced performance impact of PTI: YES (CPU supports INVPCID, performance impact of PTI will be␣
↪greatly reduced)
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: YES
> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: YES (Mitigation: Speculative Store Bypass disabled via␣
↪prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES (systemd-journald systemd-logind␣
↪systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)
> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
```

(continues on next page)

(continued from previous page)

```
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports PTE inversion: YES (found in kernel image)
* PTE inversion enabled and active: NO
> STATUS: NOT VULNERABLE (Not affected)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* Information from the /sys interface: Not affected
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: YES (found flush_l1d in /proc/cpuinfo)
  * L1D flush enabled: NO
  * Hardware-backed L1D flush supported: YES (performance impact of the mitigation will be greatly␣
↪reduced)
  * Hyper-Threading (SMT) is enabled: YES
> STATUS: NOT VULNERABLE (your kernel reported your CPU model as not vulnerable)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2019-11135 aka ZombieLoad V2, TSX Asynchronous Abort (TAA)
* Mitigated according to the /sys interface: YES (Mitigation: TSX disabled)
* TAA mitigation is supported by kernel: YES (found tsx_async_abort in kernel image)
* TAA mitigation enabled and active: YES (Mitigation: TSX disabled)
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12207 aka No eXcuses, iTLB Multihit, machine check exception on page size changes (MCEPSC)
* Mitigated according to the /sys interface: YES (KVM: Mitigation: Split huge pages)
* This system is a host running a hypervisor: NO
* iTLB Multihit mitigation is supported by kernel: YES (found itlb_multihit in kernel image)
* iTLB Multihit mitigation enabled and active: YES (KVM: Mitigation: Split huge pages)
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:OK CVE-2018-3639:OK CVE-
↪2018-3615:OK CVE-2018-3620:OK CVE-2018-3646:OK CVE-2018-12126:OK CVE-2018-12130:OK CVE-2018-
↪12127:OK CVE-2019-11091:OK CVE-2019-11135:OK CVE-2018-12207:OK
```

(continues on next page)

(continued from previous page)

```
```

2.6.6 Calibration Data - Haswell

Following sections include sample calibration data measured on t1-sut1 server running in one of the Intel Xeon Haswell testbeds as specified in FD.io CSIT Testbeds - Xeon Haswell[66].

Calibration data obtained from all other servers in Haswell testbeds shows the same or similar values.

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.15.0-72-generic root=UUID=c59ae603-8076-41f4-bb5d-bc3fc8dd3ea1 ro isolcpus=1-
↪17,19-35 nohz_full=1-17,19-35 rcu_nocbs=1-17,19-35 numa_balancing=disable intel_pstate=disable␣
↪intel_iommu=on iommu=pt nmi_watchdog=0 audit=0 nosoftlockup processor.max_cstate=1 intel_idle.max_
↪cstate=1 hpet=disable tsc=reliable mce=off console=tty0console=ttyS0,115200n8
```

Linux uname

```
$ uname -a
Linux t1-tg1 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64 x86_64 x86_64 GNU/
↪Linux
```

System-level Core Jitter

```
$ sudo taskset -c 3 /home/testuser/pma_tools/jitter/jitter -i 30
Linux Jitter testing program version 1.8
Iterations=30
The pragram will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Timings are in CPU Core cycles
Inst_Min:    Minimum Excution time during the display update interval(default is ~1 second)
Inst_Max:    Maximum Excution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Excution time during rhe display update interval. This is the value of␣
↪interest
last_Exec:   The Excution time of last iteration just before the display update
Abs_Min:     Absolute Minimum Excution time since the program started or statistics were reset
Abs_Max:     Absolute Maximum Excution time since the program started or statistics were reset
tmp:         Cumulative value calcualted by the dummy function
Interval:    Time interval between the display updates in Core Cycles
Sample No:   Sample number

   Inst_Min   Inst_Max   Inst_jitter last_Exec  Abs_min    Abs_max      tmp       Interval     ␣
↪Sample No
    160024     172636      12612     160028     160024     172636    1573060608 3205463144         ␣
↪1
    160024     188236      28212     160028     160024     188236     958595072 3205500844         ␣
↪2
    160024     185676      25652     160028     160024     188236     344129536 3205485976         ␣
↪3
    160024     172608      12584     160024     160024     188236    4024631296 3205472740         ␣
↪4
    160024     179260      19236     160028     160024     188236    3410165760 3205502164         ␣
↪5
```

(continues on next page)

[66] https://git.fd.io/csit/tree/docs/lab/testbeds_ucs_hsw_hw_bios_cfg.md?h=rls1908_2

(continued from previous page)

```
    160024      172432       12408      160024      160024      188236     2795700224 3205452036          ␣
↪6
    160024      178820       18796      160024      160024      188236     2181234688 3205455408          ␣
↪7
    160024      172512       12488      160028      160024      188236     1566769152 3205461528          ␣
↪8
    160024      172636       12612      160028      160024      188236      952303616 3205478820          ␣
↪9
    160024      173676       13652      160028      160024      188236      337838080 3205470412         ␣
↪10
    160024      178776       18752      160028      160024      188236     4018339840 3205481472         ␣
↪11
    160024      172788       12764      160028      160024      188236     3403874304 3205492336         ␣
↪12
    160024      174616       14592      160028      160024      188236     2789408768 3205474904         ␣
↪13
    160024      174440       14416      160028      160024      188236     2174943232 3205479448         ␣
↪14
    160024      178748       18724      160024      160024      188236     1560477696 3205482668         ␣
↪15
    160024      172588       12564      169404      160024      188236      946012160 3205510496         ␣
↪16
    160024      172636       12612      160024      160024      188236      331546624 3205472204         ␣
↪17
    160024      172480       12456      160024      160024      188236     4012048384 3205455864         ␣
↪18
    160024      172740       12716      160028      160024      188236     3397582848 3205464932         ␣
↪19
    160024      179200       19176      160028      160024      188236     2783117312 3205476012         ␣
↪20
    160024      172480       12456      160028      160024      188236     2168651776 3205465632         ␣
↪21
    160024      172728       12704      160024      160024      188236     1554186240 3205497204         ␣
↪22
    160024      172620       12596      160028      160024      188236      939720704 3205466972         ␣
↪23
    160024      172640       12616      160028      160024      188236      325255168 3205471216         ␣
↪24
    160024      172484       12460      160028      160024      188236     4005756928 3205467388         ␣
↪25
    160024      172636       12612      160028      160024      188236     3391291392 3205482748         ␣
↪26
    160024      179056       19032      160024      160024      188236     2776825856 3205467152         ␣
↪27
    160024      172672       12648      160024      160024      188236     2162360320 3205483268         ␣
↪28
    160024      176932       16908      160024      160024      188236     1547894784 3205488536         ␣
↪29
    160024      172452       12428      160028      160024      188236      933429248 3205440636         ␣
↪30
```

Memory Bandwidth

```
$ sudo /home/testuser/mlc --bandwidth_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --bandwidth_matrix

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes
Measuring Memory Bandwidths between nodes within system
```

(continues on next page)

(continued from previous page)

```
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
                 Numa node
Numa node        0       1
    0        57935.5   30265.2
    1        30284.6   58409.9
```

```
$ sudo /home/testuser/mlc --peak_injection_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --peak_injection_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Peak Injection Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :  115762.2
3:1 Reads-Writes :  106242.2
2:1 Reads-Writes :  103031.8
1:1 Reads-Writes :  87943.7
Stream-triad like:  100048.4
```

```
$ sudo /home/testuser/mlc --max_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --max_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Maximum Memory Bandwidths for the system
Will take several minutes to complete as multiple injection rates will be tried to get the best␣
↪bandwidth
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :  115782.41
3:1 Reads-Writes :  105965.78
2:1 Reads-Writes :  103162.38
1:1 Reads-Writes :  88255.82
Stream-triad like:  105608.10
```

Memory Latency

```
$ sudo /home/testuser/mlc --latency_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --latency_matrix

Using buffer size of 200.000MB
Measuring idle latencies (in ns)...
                 Numa node
Numa node        0       1
    0           101.0   132.0
    1           141.2    98.8
```

```
$ sudo /home/testuser/mlc --idle_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --idle_latency
```

(continues on next page)

(continued from previous page)

```

Using buffer size of 200.000MB
Each iteration took 227.2 core clocks ( 99.0    ns)
```

```
$ sudo /home/testuser/mlc --loaded_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --loaded_latency

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Loaded Latencies for the system
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  294.08   115841.6
 00002  294.27   115851.5
 00008  293.67   115821.8
 00015  278.92   115587.5
 00050  246.80   113991.2
 00100  206.86   104508.1
 00200  123.72    72873.6
 00300  113.35    52641.1
 00400  108.89    41078.9
 00500  108.11    33699.1
 00700  106.19    24878.0
 01000  104.75    17948.1
 01300  103.72    14089.0
 01700  102.95    11013.6
 02500  102.25     7756.3
 03500  101.81     5749.3
 05000  101.46     4230.4
 09000  101.05     2641.4
 20000  100.77     1542.5
```

L1/L2/LLC Latency

```
$ sudo /home/testuser/mlc --c2c_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --c2c_latency

Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency    42.1
Local Socket L2->L2 HITM latency    47.0
Remote Socket L2->L2 HITM latency (data address homed in writer socket)
                  Reader Numa Node
Writer Numa Node     0       1
            0        -   108.0
            1    106.9       -
Remote Socket L2->L2 HITM latency (data address homed in reader socket)
                  Reader Numa Node
Writer Numa Node     0       1
            0        -   107.7
            1    106.6       -
```

Spectre and Meltdown Checks

Following section displays the output of a running shell script to tell if system is vulnerable against the several "speculative execution" CVEs that were made public in 2018. Script is available on Spectre & Meltdown Checker Github[67].

```
Spectre and Meltdown mitigation detection tool v0.43

awk: cannot open bash (No such file or directory)
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26 12:20:02 UTC 2019 x86_64
CPU is Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: YES (Intel SSBD)
  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: YES
    * CPU indicates L1D flush capability: YES (L1D flush feature bit)
  * Microarchitectural Data Sampling
    * VERW instruction is available: YES (MD_CLEAR feature bit)
  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: NO
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: NO
  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): NO
  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: NO
  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): NO
  * CPU explicitly indicates not being vulnerable to TSX Asynchronous Abort (TAA_NO): NO
  * CPU explicitly indicates not being vulnerable to iTLB Multihit (PSCHANGE_MSC_NO): NO
  * CPU explicitly indicates having MSR for TSX control (TSX_CTRL_MSR): NO
  * CPU supports Transactional Synchronization Extensions (TSX): NO
  * CPU supports Software Guard Extensions (SGX): NO
  * CPU microcode is known to cause stability problems: NO (model 0x3f family 0x6 stepping 0x2␣
↪ucode 0x43 cpuid 0x306f2)
  * CPU microcode is the latest known available version: awk: cannot open bash (No such file or␣
↪directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): YES
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): YES
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): YES
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪YES
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling␣
↪(MFBDS)): YES
```

(continues on next page)

[67] https://github.com/speed47/spectre-meltdown-checker

(continued from previous page)

```
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): YES
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): YES
  * Vulnerable to CVE-2019-11135 (ZombieLoad V2, TSX Asynchronous Abort (TAA)): NO
  * Vulnerable to CVE-2018-12207 (No eXcuses, iTLB Multihit, machine check exception on page size␣
↪changes (MCEPSC)): YES

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: usercopy/swapgs barriers and __user␣
↪pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: usercopy/swapgs barriers and __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Full generic retpoline, IBPB:␣
↪conditional, IBRS_FW, RSB filling)
* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES (for firmware code only)
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES
* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
    * Kernel compiled with a retpoline-aware compiler: YES (kernel reports full retpoline␣
↪compilation)
> STATUS: NOT VULNERABLE (Full retpoline + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Mitigation: PTI)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: YES
  * Reduced performance impact of PTI: YES (CPU supports INVPCID, performance impact of PTI will be␣
↪greatly reduced)
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (Mitigation: PTI)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: YES
> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: YES (Mitigation: Speculative Store Bypass disabled via␣
↪prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES (systemd-journald systemd-logind␣
↪systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)
> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
* Mitigated according to the /sys interface: YES (Mitigation: PTE Inversion; VMX: conditional cache␣
↪flushes, SMT disabled)
* Kernel supports PTE inversion: YES (found in kernel image)
```

(continues on next page)

(continued from previous page)

```
* PTE inversion enabled and active: YES
> STATUS: NOT VULNERABLE (Mitigation: PTE Inversion; VMX: conditional cache flushes, SMT disabled)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* Information from the /sys interface: Mitigation: PTE Inversion; VMX: conditional cache flushes,␣
↪SMT disabled
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: YES (found flush_l1d in /proc/cpuinfo)
  * L1D flush enabled: YES (conditional flushes)
  * Hardware-backed L1D flush supported: YES (performance impact of the mitigation will be greatly␣
↪reduced)
  * Hyper-Threading (SMT) is enabled: NO
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT disabled)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: YES
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT disabled)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: YES
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT disabled)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: YES
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Mitigated according to the /sys interface: YES (Mitigation: Clear CPU buffers; SMT disabled)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: YES
* SMT is either mitigated or disabled: YES
> STATUS: NOT VULNERABLE (Your microcode and kernel are both up to date for this mitigation, and␣
↪mitigation is enabled)

CVE-2019-11135 aka ZombieLoad V2, TSX Asynchronous Abort (TAA)
* Mitigated according to the /sys interface: YES (Not affected)
* TAA mitigation is supported by kernel: YES (found tsx_async_abort in kernel image)
* TAA mitigation enabled and active: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12207 aka No eXcuses, iTLB Multihit, machine check exception on page size changes (MCEPSC)
* Mitigated according to the /sys interface: YES (KVM: Mitigation: Split huge pages)
* This system is a host running a hypervisor: NO
* iTLB Multihit mitigation is supported by kernel: YES (found itlb_multihit in kernel image)
* iTLB Multihit mitigation enabled and active: YES (KVM: Mitigation: Split huge pages)
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)
```

(continues on next page)

(continued from previous page)

```
> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:OK CVE-2018-3639:OK CVE-
↪2018-3615:OK CVE-2018-3620:OK CVE-2018-3646:OK CVE-2018-12126:OK CVE-2018-12130:OK CVE-2018-
↪12127:OK CVE-2019-11091:OK CVE-2019-11135:OK CVE-2018-12207:OK
```

2.6.7 Calibration Data - Denverton

Following sections include sample calibration data measured on Denverton server at Intel SH labs.

A 2-Node Atom Denverton testing took place at Intel Corporation carefully adhering to FD.io CSIT best practices.

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-36-generic root=UUID=d3cfffd0-1e77-423a-a53a-a117199b6025 ro intel_
↪iommu=on iommu=pt isolcpus=1-11 nohz_full=1-11 rcu_nocbs=1-11 default_hugepagesz=1G hugepagesz=1G␣
↪hugepages=8 intel_pstate=disable nmi_watchdog=0 numa_balancing=disable tsc=reliable nosoftlockup␣
↪quiet splash vt.handoff=7
```

Linux uname

```
$ uname -a
Linux 4.15.0-36-generic #39~16.04.1-Ubuntu SMP Tue Sep 25 08:59:23 UTC 2018 x86_64 x86_64 x86_64␣
↪GNU/Linux
```

System-level Core Jitter

```
$ sudo taskset -c 2 /home/testuser/pma_tools/jitter/jitter -c 2 -i 20
Linux Jitter testing program version 1.9
Iterations=20
The pragram will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Thread affinity will be set to core_id:2
Timings are in CPU Core cycles
Inst_Min:    Minimum Excution time during the display update interval(default is ~1 second)
Inst_Max:    Maximum Excution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Excution time during rhe display update interval. This is the value of␣
↪interest
last_Exec:   The Excution time of last iteration just before the display update
Abs_Min:     Absolute Minimum Excution time since the program started or statistics were reset
Abs_Max:     Absolute Maximum Excution time since the program started or statistics were reset
tmp:         Cumulative value calcualted by the dummy function
Interval:    Time interval between the display updates in Core Cycles
Sample No:   Sample number

   Inst_Min   Inst_Max   Inst_jitter last_Exec  Abs_min    Abs_max      tmp       Interval     ␣
↪Sample No
    177530     196100       18570     177530     177530     196100    4156751872 3556820054         ␣
↪1
    177530     200784       23254     177530     177530     200784     321060864 3556897644         ␣
↪2
    177530     196346       18816     177530     177530     200784     780337152 3556918674         ␣
↪3
```

(continues on next page)

(continued from previous page)

```
   177530     195962      18432     177530     177530     200784    1239613440 3556847928          ␣
↪4
   177530     195960      18430     177530     177530     200784    1698889728 3556860214          ␣
↪5
   177530     198824      21294     177530     177530     200784    2158166016 3556854934          ␣
↪6
   177530     198522      20992     177530     177530     200784    2617442304 3556862410          ␣
↪7
   177530     196362      18832     177530     177530     200784    3076718592 3556851636          ␣
↪8
   177530     199114      21584     177530     177530     200784    3535994880 3556870846          ␣
↪9
   177530     197194      19664     177530     177530     200784    3995271168 3556933584         ␣
↪10
   177530     198272      20742     177536     177530     200784     159580160 3556869044         ␣
↪11
   177530     197586      20056     177530     177530     200784     618856448 3556903482         ␣
↪12
   177530     196072      18542     177530     177530     200784    1078132736 3556825540         ␣
↪13
   177530     196354      18824     177530     177530     200784    1537409024 3556881664         ␣
↪14
   177530     195906      18376     177530     177530     200784    1996685312 3556839924         ␣
↪15
   177530     199066      21536     177530     177530     200784    2455961600 3556860220         ␣
↪16
   177530     196968      19438     177530     177530     200784    2915237888 3556871890         ␣
↪17
   177530     195896      18366     177530     177530     200784    3374514176 3556855338         ␣
↪18
   177530     196020      18490     177530     177530     200784    3833790464 3556839820         ␣
↪19
   177530     196030      18500     177530     177530     200784    4293066752 3556889196         ␣
↪20
```

Memory Bandwidth

```
$ sudo /home/testuser/mlc --bandwidth_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --bandwidth_matrix

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes
Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
        Memory node
Socket       0
     0  28157.2
```

```
$ sudo /home/testuser/mlc --peak_injection_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --peak_injection_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Peak Injection Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
```

(continues on next page)

(continued from previous page)

```
Using traffic with the following read-write ratios
ALL Reads        :      28150.0
3:1 Reads-Writes :      27425.0
2:1 Reads-Writes :      27565.4
1:1 Reads-Writes :      27489.3
Stream-triad like:      26878.2
```

```
$ sudo /home/testuser/mlc --max_bandwidth
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --max_bandwidth

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Maximum Memory Bandwidths for the system
Will take several minutes to complete as multiple injection rates will be tried to get the best␣
↪bandwidth
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :      30032.40
3:1 Reads-Writes :      27450.88
2:1 Reads-Writes :      27567.46
1:1 Reads-Writes :      27501.90
Stream-triad like:      27124.82
```

Memory Latency

```
$ sudo /home/testuser/mlc --latency_matrix
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --latency_matrix

Using buffer size of 2000.000MB
Intel(R) Memory Latency Checker - v3.5
Measuring idle latencies (in ns)...
        Memory node
Socket       0
     0    93.1
```

```
$ sudo /home/testuser/mlc --idle_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --idle_latency

Using buffer size of 200.000MB
Each iteration took 186.7 core clocks ( 93.4    ns)
```

```
$ sudo /home/testuser/mlc --loaded_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --loaded_latency

Using buffer size of 100.000MB/thread for reads and an additional 100.000MB/thread for writes

Measuring Loaded Latencies for the system
Using all the threads from each core if Hyper-threading is enabled
Using Read-only traffic type
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  135.35    27186.0
```

(continues on next page)

(continued from previous page)

```
00002  135.47    27176.9
00008  134.97    27063.3
00015  134.41    26825.6
00050  139.83    28419.1
00100  124.28    22616.4
00200  109.40    14139.8
00300  104.56    10275.1
00400  102.02     8120.0
00500  100.38     6751.4
00700   98.30     5124.9
01000   96.56     3852.7
01300   95.65     3149.0
01700   95.06     2585.4
02500   94.43     1988.8
03500   94.16     1621.1
05000   93.95     1343.1
09000   93.65     1052.6
20000   93.43      851.7
```

L1/L2/LLC Latency

```
$ sudo /home/testuser/mlc --c2c_latency
Intel(R) Memory Latency Checker - v3.5
Command line parameters: --c2c_latency

Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency        8.8
Local Socket L2->L2 HITM latency        8.8
```

Spectre and Meltdown Checks

Following section displays the output of a running shell script to tell if system is vulnerable against the several "speculative execution" CVEs that were made public in 2018. Script is available on Spectre & Meltdown Checker Github[68].

```
Spectre and Meltdown mitigation detection tool v0.42
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-51-generic #55-Ubuntu SMP Wed May 15 14:27:21 UTC 2019 x86_64
CPU is Intel(R) Atom(TM) CPU C3858 @ 2.00GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: YES (Intel SSBD)
  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: NO
    * CPU indicates L1D flush capability: NO
```

(continues on next page)

[68] https://github.com/speed47/spectre-meltdown-checker

(continued from previous page)

```
  * Microarchitecture Data Sampling
    * VERW instruction is available: YES (MD_CLEAR feature bit)
  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: YES
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: NO
  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): YES
  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: YES
  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): YES
  * CPU supports Software Guard Extensions (SGX): NO
  * CPU microcode is known to cause stability problems: NO (model 0x5f family 0x6 stepping 0x1␣
↪ucode 0x2e cpuid 0x506f1)
  * CPU microcode is the latest known available version: awk: fatal: cannot open file `bash for␣
↪reading (No such file or directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): NO
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): NO
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪NO
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling␣
↪(MFBDS)): NO
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): NO
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): NO

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: __user pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Full generic retpoline, IBPB:␣
↪conditional, IBRS_FW, STIBP: disabled, RSB filling)
* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES (for firmware code only)
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES
* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
    * Kernel compiled with a retpoline-aware compiler: YES (kernel reports full retpoline␣
↪compilation)
> STATUS: NOT VULNERABLE (Full retpoline + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: UNKNOWN (dmesg truncated, please reboot and relaunch this script)
  * Reduced performance impact of PTI: NO (PCID/INVPCID not supported, performance impact of PTI␣
↪will be significant)
```

(continues on next page)

(continued from previous page)

```
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: YES
> STATUS: NOT VULNERABLE (your CPU microcode mitigates the vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: YES (Mitigation: Speculative Store Bypass disabled via␣
↪prctl and seccomp)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: YES (per-thread through prctl)
* SSB mitigation currently active for selected processes: YES (systemd-journald systemd-logind␣
↪systemd-networkd systemd-resolved systemd-timesyncd systemd-udevd)
> STATUS: NOT VULNERABLE (Mitigation: Speculative Store Bypass disabled via prctl and seccomp)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports PTE inversion: YES (found in kernel image)
* PTE inversion enabled and active: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* Information from the /sys interface: Not affected
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: YES (found flush_l1d in kernel image)
  * L1D flush enabled: NO
  * Hardware-backed L1D flush supported: NO (flush will be done in software, this is slower)
  * Hyper-Threading (SMT) is enabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Mitigated according to the /sys interface: YES (Not affected)
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Mitigated according to the /sys interface: YES (Not affected)
```

(continues on next page)

(continued from previous page)

```
* Kernel supports using MD_CLEAR mitigation: YES (md_clear found in /proc/cpuinfo)
* Kernel mitigation is enabled and active: NO
* SMT is either mitigated or disabled: NO
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:OK CVE-2018-3639:OK CVE-
↪2018-3615:OK CVE-2018-3620:OK CVE-2018-3646:OK CVE-2018-12126:OK CVE-2018-12130:OK CVE-2018-
↪12127:OK CVE-2019-11091:OK
```

2.6.8 Calibration Data - TaiShan

Following sections include sample calibration data measured on s17-t33-sut1 server running in one of the Cortex-A72 testbeds.

Calibration data obtained from all other servers in TaiShan testbeds shows the same or similar values.

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.15.0-54-generic root=/dev/mapper/huawei--1--vg-root ro isolcpus=1-15,17-
↪31,33-47,49-63 nohz_full=1-15     17-31,33-47,49-63 rcu_nocbs=1-15     17-31,33-47,49-63 intel_
↪iommu=on nmi_watchdog=0 audit=0 nosoftlockup processor.max_cstate=1 console=ttyAMA0,115200n8
```

Linux uname

```
$ uname -a
Linux s17-t33-sut1 4.15.0-54-generic #58-Ubuntu SMP Mon Jun 24 10:56:40 UTC 2019 aarch64 aarch64␣
↪aarch64 GNU/Linux
```

System-level Core Jitter

```
$ sudo taskset -c 3 /home/testuser/pma_tools/jitter/jitter -i 20
Linux Jitter testing program version 1.9
Iterations=30
The pragram will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Thread affinity will be set to core_id:7
Timings are in CPU Core cycles
Inst_Min:    Minimum Excution time during the display update interval(default is ~1 second)
Inst_Max:    Maximum Excution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Excution time during rhe display update interval. This is the value of␣
↪interest
last_Exec:   The Excution time of last iteration just before the display update
Abs_Min:     Absolute Minimum Excution time since the program started or statistics were reset
Abs_Max:     Absolute Maximum Excution time since the program started or statistics were reset
tmp:         Cumulative value calcualted by the dummy function
Interval:    Time interval between the display updates in Core Cycles
Sample No:   Sample number

   Inst_Min   Inst_Max   Inst_jitter last_Exec  Abs_min    Abs_max      tmp       Interval     ␣
↪Sample No
    160022     172254      12232      160042     160022     172254    1903230976 3204401362          ␣
↪1
    160022     173148      13126      160044     160022     173148     814809088 3204619316          ␣
↪2
```

(continues on next page)

(continued from previous page)

```
    160022      169460       9438     160044     160022     173148    4021354496  3204391306           ␣
↪3
    160024      170270      10246     160044     160022     173148    2932932608  3204385830           ␣
↪4
    160022      169660       9638     160044     160022     173148    1844510720  3204387290           ␣
↪5
    160022      169410       9388     160040     160022     173148     756088832  3204375832           ␣
↪6
    160022      169012       8990     160042     160022     173148    3962634240  3204378924           ␣
↪7
    160022      169556       9534     160044     160022     173148    2874212352  3204374882           ␣
↪8
    160022      171684      11662     160042     160022     173148    1785790464  3204394596           ␣
↪9
    160022      171546      11524     160024     160022     173148     697368576  3204602774          ␣
↪10
    160022      169248       9226     160042     160022     173148    3903913984  3204401676          ␣
↪11
    160022      168458       8436     160042     160022     173148    2815492096  3204256350          ␣
↪12
    160022      169574       9552     160044     160022     173148    1727070208  3204278116          ␣
↪13
    160022      169352       9330     160044     160022     173148     638648320  3204327234          ␣
↪14
    160022      169100       9078     160044     160022     173148    3845193728  3204388132          ␣
↪15
    160022      169338       9316     160042     160022     173148    2756771840  3204380724          ␣
↪16
    160022      170828      10806     160046     160022     173148    1668349952  3204430452          ␣
↪17
    160022      173162      13140     160026     160022     173162     579928064  3204611318          ␣
↪18
    160022      170482      10460     160042     160022     173162    3786473472  3204389896          ␣
↪19
    160024      170704      10680     160044     160022     173162    2698051584  3204422126          ␣
↪20
    160024      169302       9278     160044     160022     173162    1609629696  3204397334          ␣
↪21
    160022      171848      11826     160044     160022     173162     521207808  3204389818          ␣
↪22
    160022      169438       9416     160042     160022     173162    3727753216  3204395382          ␣
↪23
    160022      169312       9290     160042     160022     173162    2639331328  3204371202          ␣
↪24
    160022      171368      11346     160044     160022     173162    1550909440  3204440464          ␣
↪25
    160022      171998      11976     160042     160022     173162     462487552  3204609440          ␣
↪26
    160022      169740       9718     160046     160022     173162    3669032960  3204405826          ␣
↪27
    160022      169610       9588     160044     160022     173162    2580611072  3204390608          ␣
↪28
    160022      169254       9232     160044     160022     173162    1492189184  3204399760          ␣
↪29
    160022      169386       9364     160046     160022     173162     403767296  3204417762          ␣
↪30
```

Spectre and Meltdown Checks

Following section displays the output of a running shell script to tell if system is vulnerable against the several “speculative execution” CVEs that were made public in 2018. Script is available on Spectre & Meltdown Checker Github[69].

```
Spectre and Meltdown mitigation detection tool v0.43

awk: cannot open bash (No such file or directory)
Checking for vulnerabilities on current system
Kernel is Linux 4.15.0-23-generic #25-Ubuntu SMP Wed May 23 18:02:16 UTC 2018 x86_64
CPU is Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz

Hardware check
* Hardware support (CPU microcode) for mitigation techniques
  * Indirect Branch Restricted Speculation (IBRS)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates IBRS capability: YES (SPEC_CTRL feature bit)
  * Indirect Branch Prediction Barrier (IBPB)
    * PRED_CMD MSR is available: YES
    * CPU indicates IBPB capability: YES (SPEC_CTRL feature bit)
  * Single Thread Indirect Branch Predictors (STIBP)
    * SPEC_CTRL MSR is available: YES
    * CPU indicates STIBP capability: YES (Intel STIBP feature bit)
  * Speculative Store Bypass Disable (SSBD)
    * CPU indicates SSBD capability: NO
  * L1 data cache invalidation
    * FLUSH_CMD MSR is available: NO
    * CPU indicates L1D flush capability: NO
  * Microarchitectural Data Sampling
    * VERW instruction is available: NO
  * Enhanced IBRS (IBRS_ALL)
    * CPU indicates ARCH_CAPABILITIES MSR availability: NO
    * ARCH_CAPABILITIES MSR advertises IBRS_ALL capability: NO
  * CPU explicitly indicates not being vulnerable to Meltdown/L1TF (RDCL_NO): NO
  * CPU explicitly indicates not being vulnerable to Variant 4 (SSB_NO): NO
  * CPU/Hypervisor indicates L1D flushing is not necessary on this system: NO
  * Hypervisor indicates host CPU might be vulnerable to RSB underflow (RSBA): NO
  * CPU explicitly indicates not being vulnerable to Microarchitectural Data Sampling (MDS_NO): NO
  * CPU explicitly indicates not being vulnerable to TSX Asynchronous Abort (TAA_NO): NO
  * CPU explicitly indicates not being vulnerable to iTLB Multihit (PSCHANGE_MSC_NO): NO
  * CPU explicitly indicates having MSR for TSX control (TSX_CTRL_MSR): NO
  * CPU supports Transactional Synchronization Extensions (TSX): YES (RTM feature bit)
  * CPU supports Software Guard Extensions (SGX): NO
  * CPU microcode is known to cause stability problems: NO (model 0x55 family 0x6 stepping 0x4␣
↪ucode 0x2000043 cpuid 0x50654)
  * CPU microcode is the latest known available version: awk: cannot open bash (No such file or␣
↪directory)
UNKNOWN (latest microcode version for your CPU model is unknown)
* CPU vulnerability to the speculative execution attack variants
  * Vulnerable to CVE-2017-5753 (Spectre Variant 1, bounds check bypass): YES
  * Vulnerable to CVE-2017-5715 (Spectre Variant 2, branch target injection): YES
  * Vulnerable to CVE-2017-5754 (Variant 3, Meltdown, rogue data cache load): YES
  * Vulnerable to CVE-2018-3640 (Variant 3a, rogue system register read): YES
  * Vulnerable to CVE-2018-3639 (Variant 4, speculative store bypass): YES
  * Vulnerable to CVE-2018-3615 (Foreshadow (SGX), L1 terminal fault): NO
  * Vulnerable to CVE-2018-3620 (Foreshadow-NG (OS), L1 terminal fault): YES
  * Vulnerable to CVE-2018-3646 (Foreshadow-NG (VMM), L1 terminal fault): YES
  * Vulnerable to CVE-2018-12126 (Fallout, microarchitectural store buffer data sampling (MSBDS)):␣
↪YES
  * Vulnerable to CVE-2018-12130 (ZombieLoad, microarchitectural fill buffer data sampling␣
↪(MFBDS)): YES
```

(continues on next page)

[69] https://github.com/speed47/spectre-meltdown-checker

(continued from previous page)

```
  * Vulnerable to CVE-2018-12127 (RIDL, microarchitectural load port data sampling (MLPDS)): YES
  * Vulnerable to CVE-2019-11091 (RIDL, microarchitectural data sampling uncacheable memory␣
↪(MDSUM)): YES
  * Vulnerable to CVE-2019-11135 (ZombieLoad V2, TSX Asynchronous Abort (TAA)): YES
  * Vulnerable to CVE-2018-12207 (No eXcuses, iTLB Multihit, machine check exception on page size␣
↪changes (MCEPSC)): YES

CVE-2017-5753 aka Spectre Variant 1, bounds check bypass
* Mitigated according to the /sys interface: YES (Mitigation: __user pointer sanitization)
* Kernel has array_index_mask_nospec: YES (1 occurrence(s) found of x86 64 bits array_index_mask_
↪nospec())
* Kernel has the Red Hat/Ubuntu patch: NO
* Kernel has mask_nospec64 (arm64): NO
> STATUS: NOT VULNERABLE (Mitigation: __user pointer sanitization)

CVE-2017-5715 aka Spectre Variant 2, branch target injection
* Mitigated according to the /sys interface: YES (Mitigation: Full generic retpoline, IBPB, IBRS_FW)
* Mitigation 1
  * Kernel is compiled with IBRS support: YES
    * IBRS enabled and active: YES (for firmware code only)
  * Kernel is compiled with IBPB support: YES
    * IBPB enabled and active: YES
* Mitigation 2
  * Kernel has branch predictor hardening (arm): NO
  * Kernel compiled with retpoline option: YES
    * Kernel compiled with a retpoline-aware compiler: YES (kernel reports full retpoline␣
↪compilation)
  * Kernel supports RSB filling: YES
> STATUS: NOT VULNERABLE (Full retpoline + IBPB are mitigating the vulnerability)

CVE-2017-5754 aka Variant 3, Meltdown, rogue data cache load
* Mitigated according to the /sys interface: YES (Mitigation: PTI)
* Kernel supports Page Table Isolation (PTI): YES
  * PTI enabled and active: YES
  * Reduced performance impact of PTI: YES (CPU supports INVPCID, performance impact of PTI will be␣
↪greatly reduced)
* Running as a Xen PV DomU: NO
> STATUS: NOT VULNERABLE (Mitigation: PTI)

CVE-2018-3640 aka Variant 3a, rogue system register read
* CPU microcode mitigates the vulnerability: NO
> STATUS: VULNERABLE (an up-to-date CPU microcode is needed to mitigate this vulnerability)

CVE-2018-3639 aka Variant 4, speculative store bypass
* Mitigated according to the /sys interface: NO (Vulnerable)
* Kernel supports disabling speculative store bypass (SSB): YES (found in /proc/self/status)
* SSB mitigation is enabled and active: NO
> STATUS: VULNERABLE (Your CPU doesnt support SSBD)

CVE-2018-3615 aka Foreshadow (SGX), L1 terminal fault
* CPU microcode mitigates the vulnerability: N/A
> STATUS: NOT VULNERABLE (your CPU vendor reported your CPU model as not vulnerable)

CVE-2018-3620 aka Foreshadow-NG (OS), L1 terminal fault
* Kernel supports PTE inversion: NO
* PTE inversion enabled and active: UNKNOWN (sysfs interface not available)
> STATUS: VULNERABLE (Your kernel doesnt support PTE inversion, update it)

CVE-2018-3646 aka Foreshadow-NG (VMM), L1 terminal fault
* This system is a host running a hypervisor: NO
* Mitigation 1 (KVM)
```

(continues on next page)

(continued from previous page)

```
  * EPT is disabled: NO
* Mitigation 2
  * L1D flush is supported by kernel: NO
  * L1D flush enabled: UNKNOWN (cant find or read /sys/devices/system/cpu/vulnerabilities/l1tf)
  * Hardware-backed L1D flush supported: NO (flush will be done in software, this is slower)
  * Hyper-Threading (SMT) is enabled: YES
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

CVE-2018-12126 aka Fallout, microarchitectural store buffer data sampling (MSBDS)
* Kernel supports using MD_CLEAR mitigation: NO
> STATUS: VULNERABLE (Neither your kernel or your microcode support mitigation, upgrade both to␣
↪mitigate the vulnerability)

CVE-2018-12130 aka ZombieLoad, microarchitectural fill buffer data sampling (MFBDS)
* Kernel supports using MD_CLEAR mitigation: NO
> STATUS: VULNERABLE (Neither your kernel or your microcode support mitigation, upgrade both to␣
↪mitigate the vulnerability)

CVE-2018-12127 aka RIDL, microarchitectural load port data sampling (MLPDS)
* Kernel supports using MD_CLEAR mitigation: NO
> STATUS: VULNERABLE (Neither your kernel or your microcode support mitigation, upgrade both to␣
↪mitigate the vulnerability)

CVE-2019-11091 aka RIDL, microarchitectural data sampling uncacheable memory (MDSUM)
* Kernel supports using MD_CLEAR mitigation: NO
> STATUS: VULNERABLE (Neither your kernel or your microcode support mitigation, upgrade both to␣
↪mitigate the vulnerability)

CVE-2019-11135 aka ZombieLoad V2, TSX Asynchronous Abort (TAA)
* TAA mitigation is supported by kernel: NO
* TAA mitigation enabled and active: NO (tsx_async_abort not found in sysfs hierarchy)
> STATUS: VULNERABLE (Your kernel doesnt support TAA mitigation, update it)

CVE-2018-12207 aka No eXcuses, iTLB Multihit, machine check exception on page size changes (MCEPSC)
* This system is a host running a hypervisor: NO
* iTLB Multihit mitigation is supported by kernel: NO
* iTLB Multihit mitigation enabled and active: NO (itlb_multihit not found in sysfs hierarchy)
> STATUS: NOT VULNERABLE (this system is not running a hypervisor)

> SUMMARY: CVE-2017-5753:OK CVE-2017-5715:OK CVE-2017-5754:OK CVE-2018-3640:KO CVE-2018-3639:KO CVE-
↪2018-3615:OK CVE-2018-3620:KO CVE-2018-3646:OK CVE-2018-12126:KO CVE-2018-12130:KO CVE-2018-
↪12127:KO CVE-2019-11091:KO CVE-2019-11135:KO CVE-2018-12207:OK
```

2.6.9 SUT Settings - Linux

System provisioning is done by combination of PXE boot unattented install and Ansible[70] described in CSIT Testbed Setup[71].

Below a subset of the running configuration:

1. Ubuntu 18.04.x LTS

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.3 LTS
Release:        18.04
Codename:       bionic
```

[70] https://www.ansible.com
[71] https://git.fd.io/csit/tree/resources/tools/testbed-setup/README.md?h=rls1908_2

Linux Boot Parameters

- **isolcpus=<cpu number>-<cpu number>** used for all cpu cores apart from first core of each socket used for running VPP worker threads and Qemu/LXC processes https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt
- **intel_pstate=disable** - [X86] Do not enable intel_pstate as the default scaling driver for the supported processors. Intel P-State driver decide what P-state (CPU core power state) to use based on requesting policy from the cpufreq core. [X86 - Either 32-bit or 64-bit x86] https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt
- **nohz_full=<cpu number>-<cpu number>** - [KNL,BOOT] In kernels built with CONFIG_NO_HZ_FULL=y, set the specified list of CPUs whose tick will be stopped whenever possible. The boot CPU will be forced outside the range to maintain the timekeeping. The CPUs in this range must also be included in the rcu_nocbs= set. Specifies the adaptive-ticks CPU cores, causing kernel to avoid sending scheduling-clock interrupts to listed cores as long as they have a single runnable task. [KNL - Is a kernel start-up parameter, SMP - The kernel is an SMP kernel]. https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt
- **rcu_nocbs** - [KNL] In kernels built with CONFIG_RCU_NOCB_CPU=y, set the specified list of CPUs to be no-callback CPUs, that never queue RCU callbacks (read-copy update). https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt
- **numa_balancing=disable** - [KNL,X86] Disable automatic NUMA balancing.
- **intel_iommu=enable** - [DMAR] Enable Intel IOMMU driver (DMAR) option.
- **iommu=on, iommu=pt** - [x86, IA-64] Disable IOMMU bypass, using IOMMU for PCI devices.
- **nmi_watchdog=0** - [KNL,BUGS=X86] Debugging features for SMP kernels. Turn hardlockup detector in nmi_watchdog off.
- **nosoftlockup** - [KNL] Disable the soft-lockup detector.
- **tsc=reliable** - Disable clocksource stability checks for TSC. [x86] reliable: mark tsc clocksource as reliable, this disables clocksource verification at runtime, as well as the stability checks done at bootup. Used to enable high-resolution timer mode on older hardware, and in virtualized environment.
- **hpet=disable** - [X86-32,HPET] Disable HPET and use PIT instead.

Hugepages Configuration

Huge pages are namaged via sysctl configuration located in */etc/sysctl.d/90-csit.conf* on each testbed. Default huge page size is 2M. The exact amount of huge pages depends on testbed. All the values are defined in *Ansible inventory - hosts* files.

2.6.10 DUT Settings - VPP

VPP Version

VPP-19.08.2 release

VPP Compile Parameters

FD.io VPP compile job[72]

[72] https://jenkins.fd.io/view/vpp/job/vpp-merge-1908_2-ubuntu1804/

VPP Install Parameters

```
$ dpkg -i --force-all *vpp*
```

VPP Startup Configuration

VPP startup configuration vary per test case, with different settings for *$$CORELIST_WORKERS*, *$$NUM_RX_QUEUES*, *$$UIO_DRIVER*, *$$NUM- MBUFS* and *$$NO_MULTI_SEG* parameter. Default template is provided below:

```
ip
{
  heap-size 4G
}
statseg
{
  size 4G
}
unix
{
  cli-listen /run/vpp/cli.sock
  log /tmp/vpe.log
  nodaemon
}
socksvr {
  socket-name /run/vpp/api.sock
}
ip6
{
  heap-size 4G
  hash-buckets 2000000
}
heapsize 4G
plugins
{
  plugin default
  {
    disable
  }
  plugin dpdk_plugin.so
  {
    enable
  }
}
cpu
{
  corelist-workers $$CORELIST_WORKERS
  main-core 1
}
dpdk
{
  num-mbufs $$NUM-MBUFS
  uio-driver $$UIO_DRIVER
  $$NO_MULTI_SEG
  log-level debug
  dev default
  {
    num-rx-queues $$NUM_RX_QUEUES
  }
  no-tx-checksum-offload
```

(continues on next page)

(continued from previous page)

```
  dev $$DEV_1
  dev $$DEV_2
}
```

Description of VPP startup settings used in CSIT is provided in *Test Methodology* (page 13).

2.6.11 TG Settings - TRex

TG Version

TRex v2.73

DPDK Version

DPDK v19.05

TG Build Script Used

TRex installation[73]

TG Startup Configuration

```
$ cat /etc/trex_cfg.yaml
- version         : 2
  interfaces      : ["0000:0d:00.0","0000:0d:00.1"]
  port_info       :
    - dest_mac        :   [0x3c,0xfd,0xfe,0x9c,0xee,0xf5]
      src_mac         :   [0x3c,0xfd,0xfe,0x9c,0xee,0xf4]
    - dest_mac        :   [0x3c,0xfd,0xfe,0x9c,0xee,0xf4]
      src_mac         :   [0x3c,0xfd,0xfe,0x9c,0xee,0xf5]
```

TG Startup Command

```
$ sh -c 'cd <t-rex-install-dir>/scripts/ && sudo nohup ./t-rex-64 -i -c 7 --prefix $(hostname) --
↪hdrh > /tmp/trex.log 2>&1 &'> /dev/null
```

TG API Driver

TRex driver[74]

[73] https://git.fd.io/csit/tree/resources/tools/trex/trex_installer.sh?h=rls1908_2
[74] https://git.fd.io/csit/tree/resources/tools/trex/trex_stateless_profile.py?h=rls1908_2

2.7 Documentation

2.7.1 Container Orchestration in CSIT

Overview

Linux Containers

Linux Containers is an OS-level virtualization method for running multiple isolated Linux systems (containers) on a compute host using a single Linux kernel. Containers rely on Linux kernel cgroups functionality for controlling usage of shared system resources (i.e. CPU, memory, block I/O, network) and for namespace isolation. The latter enables complete isolation of applications' view of operating environment, including process trees, networking, user IDs and mounted file systems.

LXC (Linux Containers) combine kernel's cgroups and support for isolated namespaces to provide an isolated environment for applications. Docker does use LXC as one of its execution drivers, enabling image management and providing deployment services. More information in [lxc], [lxcnamespace] and [stgraber].

Linux containers can be of two kinds: privileged containers and unprivileged containers.

Unprivileged Containers

Running unprivileged containers is the safest way to run containers in a production environment. From LXC 1.0 one can start a full system container entirely as a user, allowing to map a range of UIDs on the host into a namespace inside of which a user with UID 0 can exist again. In other words an unprivileged container does mask the userid from the host, making it impossible to gain a root access on the host even if a user gets root in a container. With unprivileged containers, non-root users can create containers and will appear in the container as the root, but will appear as userid <non-zero> on the host. Unprivileged containers are also better suited to supporting multi-tenancy operating environments. More information in [lxcsecurity] and [stgraber].

Privileged Containers

Privileged containers do not mask UIDs, and container UID 0 is mapped to the host UID 0. Security and isolation is controlled by a good configuration of cgroup access, extensive AppArmor profile preventing the known attacks as well as container capabilities and SELinux. Here a list of applicable security control mechanisms:

- Capabilities - keep (whitelist) or drop (blacklist) Linux capabilities, [capabilities].
- Control groups - cgroups, resource bean counting, resource quotas, access restrictions, [cgroup1], [cgroup2].
- AppArmor - apparmor profiles aim to prevent any of the known ways of escaping a container or cause harm to the host, [apparmor].
- SELinux - Security Enhanced Linux is a Linux kernel security module that provides similar function to AppArmor, supporting access control security policies including United States Department of Defense-style mandatory access controls. Mandatory access controls allow an administrator of a system to define how applications and users can access different resources such as files, devices, networks and inter- process communication, [selinux].
- Seccomp - secure computing mode, enables filtering of system calls, [seccomp].

More information in [lxcsecurity] and [lxcsecfeatures].

Linux Containers in CSIT

CSIT is using Privileged Containers as the `sysfs` is mounted with RW access. Sysfs is required to be mounted as RW due to VPP accessing **`/sys/bus/pci/drivers/uio_pci_generic/unbind`**. This is not the case of unprivileged containers where `sysfs` is mounted as read-only.

Orchestrating Container Lifecycle Events

Following Linux container lifecycle events need to be addressed by an orchestration system:

1. Acquire - acquiring/downloading existing container images via **`docker pull`** or **`lxc-create -t download`**.
2. Build - building a container image from scratch or another container image via **`docker build <dockerfile/composefile>`** or customizing LXC templates in GitHub[75].
3. (Re-)Create - creating a running instance of a container application from anew, or re-creating one that failed. A.k.a. (re-)deploy via **`docker run`** or **`lxc-start`**
4. Execute - execute system operations within the container by attaching to running container. THis is done by **`lxc-attach`** or **`docker exec`**
5. Distribute - distributing pre-built container images to the compute nodes. Currently not implemented in CSIT.

Container Orchestration Systems Used in CSIT

Current CSIT testing framework integrates following Linux container orchestration mechanisms:

- LXC/Docker for complete VPP container lifecycle control.

LXC

LXC is the well-known and heavily tested low-level Linux container runtime [lxcsource], that provides a userspace interface for the Linux kernel containment features. With a powerful API and simple tools, LXC enables Linux users to easily create and manage system or application containers. LXC uses following kernel features to contain processes:

- Kernel namespaces: ipc, uts, mount, pid, network and user.
- AppArmor and SELinux security profiles.
- Seccomp policies.
- Chroot.
- Cgroups.

CSIT uses LXC runtime and LXC usertools to test VPP data plane performance in a range of virtual networking topologies.

Known Issues

- Current CSIT restriction: only single instance of lxc runtime due to the cgroup policies used in CSIT. There is plan to add the capability into code to create cgroups per container instance to address this issue. This sort of functionality is better supported in LXC 2.1 but can be done is current version as well.
- CSIT code is currently using cgroup to control the range of CPU cores the LXC container runs on. VPP thread pinning is defined vpp startup.conf.

[75] https://github.com/lxc/lxc/tree/master/templates

Docker

Docker builds on top of Linux kernel containment features, and offers a high-level tool for wrapping the processes, maintaining and executing them in containers [docker]. Currently it using *runc* a CLI tool for spawning and running containers according to the OCI specification[76]

A Docker container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

CSIT uses Docker to manage the maintenance and execution of containerized applications used in CSIT performance tests.

- Data plane thread pinning to CPU cores - Docker CLI and/or Docker configuration file controls the range of CPU cores the Docker image must run on. VPP thread pinning defined vpp startup.conf.

Implementation

CSIT container orchestration is implemented in CSIT Level-1 keyword Python libraries following the Builder design pattern. Builder design pattern separates the construction of a complex object from its representation, so that the same construction process can create different representations e.g. LXC, Docker, other.

CSIT Robot Framework keywords are then responsible for higher level lifecycle control of of the named container groups. One can have multiple named groups, with 1..N containers in a group performing different role/functionality e.g. NFs, Switch, Kafka bus, ETCD datastore, etc. ContainerManager class acts as a Director and uses ContainerEngine class that encapsulate container control.

Current CSIT implementation is illustrated using UML Class diagram:

1. Acquire
2. Build
3. (Re-)Create
4. Execute

```
+-----------------------------------------------------------------------+
|              RF Keywords (high level lifecycle control)               |
+-----------------------------------------------------------------------+
| Construct VNF containers on all DUTs                                  |
| Acquire all '${group}' containers                                     |
| Create all '${group}' containers                                      |
| Install all '${group}' containers                                     |
| Configure all '${group}' containers                                   |
| Stop all '${group}' containers                                        |
| Destroy all '${group}' containers                                     |
+-----------------+-----------------------------------------------------+
                  |  1
                  |
                  |  1..N
+-----------------v-----------------+        +--------------------------+
|          ContainerManager         |        |  ContainerEngine         |
+-----------------------------------+        +--------------------------+
| __init()__                        |        | __init(node)__           |
| construct_container()             |        | acquire(force)           |
| construct_containers()            |        | create()                 |
| acquire_all_containers()          |        | stop()                   |
| create_all_containers()           | 1    1 | destroy()                |
| execute_on_container()            <>-------| info()                   |
| execute_on_all_containers()       |        | execute(command)         |
```

(continues on next page)

[76] https://www.opencontainers.org/

(continued from previous page)

```
| install_vpp_in_all_containers()   |        | system_info()            |
| configure_vpp_in_all_containers() |        | install_supervisor()     |
| stop_all_containers()             |        | install_vpp()            |
| destroy_all_containers()          |        | restart_vpp()            |
+-----------------------------------+        | create_vpp_exec_config() |
                                             | create_vpp_startup_config|
                                             | is_container_running()   |
                                             | is_container_present()   |
                                             | _configure_cgroup()      |
                                             +-------------^------------+
                                                           |
                                                           |
                                                           |
                                                +----------+---------+
                                                |                    |
                                         +------+-------+     +------+-------+
                                         |     LXC      |     |    Docker    |
                                         +--------------+     +--------------+
                                         | (inherinted) |     | (inherinted) |
                                         +------+-------+     +------+-------+
                                                 |                   |
                                                 +---------+---------+
                                                           |
                                                           | constructs
                                                           |
                                                 +---------v---------+
                                                 |     Container     |
                                                 +-------------------+
                                                 | __getattr__(a)    |
                                                 | __setattr__(a, v) |
                                                 +-------------------+
```

Sequential diagram that illustrates the creation of a single container.

```
Legend:
   e  = engine [Docker|LXC]
   .. = kwargs (variable number of keyword argument)

+-------+                  +------------------+      +-----------------+
| RF KW |                  | ContainerManager |      | ContainerEngine |
+---+---+                  +--------+---------+      +--------+--------+
    |                               |                          |
    |  1: new ContainerManager(e)   |                          |
   +-+---------------------------->+-+                         |
   |-|                             |-| 2: new ContainerEngine  |
   |-|                             |-+----------------------->+-+
   |-|                             |-|                        |-|
   |-|                             +-+                        +-+
   |-|                              |                          |
   |-| 3: construct_container(..)   |                          |
   |-+---------------------------->+-+                         |
   |-|                             |-| 4: init()               |
   |-|                             |-+----------------------->+-+
   |-|                             |-|                        |-| 5: new  +-------------+
   |-|                             |-|                        |-+-------->| Container A |
   |-|                             |-|                        |-|         +-------------+
   |-|                             |-|<-----------------------+-|
   |-|                             +-+                        +-+
   |-|                              |                          |
   |-| 6: acquire_all_containers()  |                          |
   |-+---------------------------->+-+                         |
```

(continues on next page)

(continued from previous page)

```
   |-|                             |-| 7: acquire()            |
   |-|                             |-+----------------------->+-+
   |-|                             |-|                        |-|
   |-|                             |-|                        |-+--+
   |-|                             |-|                        |-|  | 8: is_container_present()
   |-|                             |-|             True/False |-|<-+
   |-|                             |-|                        |-|
   |-|                             |-|                        |-|
+---------------------------------------------------------------------------------------------+
|  |-| ALT [isRunning & force]     |-|                        |-|--+                          |
|  |-|                             |-|                        |-|  | 8a: destroy()            |
|  |-|                             |-|                        |-<--+                          |
+---------------------------------------------------------------------------------------------+
   |-|                             |-|                        |-|
   |-|                             +-+                        +-+
   |-|                              |                          |
   |-| 9: create_all_containers()   |                          |
   |-+---------------------------->+-+                         |
   |-|                             |-| 10: create()            |
   |-|                             |-+----------------------->+-+
   |-|                             |-|                        |-+--+
   |-|                             |-|                        |-|  | 11: wait('RUNNING')
   |-|                             |-|                        |-<--+
   |-|                             +-+                        +-+
   |-|                              |                          |
+---------------------------------------------------------------------------------------------+
|  |-| ALT                          |                          |                              |
|  |-| (install_vpp, configure_vpp) |                          |                              |
|  |-|                              |                          |                              |
+---------------------------------------------------------------------------------------------+
   |-|                              |                          |
   |-| 12: destroy_all_containers() |                          |
   |-+---------------------------->+-+                         |
   |-|                             |-| 13: destroy()           |
   |-|                             |-+----------------------->+-+
   |-|                             |-|                        |-|
   |-|                             +-+                        +-+
   |-|                              |                          |
   +++                              |                          |
    |                               |                          |
    +                               +                          +
```

Container Data Structure

Container is represented in Python L1 library as a separate Class with instance variables and no methods except overriden `__getattr__` and `__setattr__`. Instance variables are assigned to container dynamically during the `construct_container(**kwargs)` call and are passed down from the RF keyword.

Usage example:

```
| Construct VNF containers on all DUTs
| | [Arguments] | ${technology} | ${image} | ${cpu_count}=${1} | ${count}=${1}
| | ...
| | ${group}= | Set Variable | VNF
| | ${skip_cpus}= | Evaluate | ${vpp_cpus}+${system_cpus}
| | Import Library | resources.libraries.python.ContainerUtils.ContainerManager
| | ... | engine=${container_engine} | WITH NAME | ${group}
| | ${duts}= | Get Matches | ${nodes} | DUT*
| | :FOR | ${dut} | IN | @{duts}
```

(continues on next page)

(continued from previous page)

```
| | | ${env}= | Create List | DEBIAN_FRONTEND=noninteractive
| | | ${mnt}= | Create List | /tmp:/mnt/host | /dev:/dev
| | | ${cpu_node}= | Get interfaces numa node | ${nodes['${dut}']}
| | | ... | ${dut1_if1} | ${dut1_if2}
| | | Run Keyword | ${group}.Construct containers
| | | ... | name=${dut}_${group} | node=${nodes['${dut}']} | mnt=${mnt}
| | | ... | image=${container_image} | cpu_count=${container_cpus}
| | | ... | cpu_skip=${skip_cpus} | cpuset_mems=${cpu_node}
| | | ... | cpu_shared=${False} | env=${env} | count=${container_count}
| | | ... | install_dkms=${container_install_dkms}
| | Append To List | ${container_groups} | ${group}
```

Mandatory parameters to create standalone container are: `node`, `name`, `image` [imagevar], `cpu_count`, `cpu_skip`, `cpuset_mems`, `cpu_shared`.

There is no parameters check functionality. Passing required arguments is in coder responsibility. All the above parameters are required to calculate the correct cpu placement. See documentation for the full reference.

Kubernetes

For the future use, Kubernetes [k8sdoc] is implemented as separate library `KubernetesUtils.py`, with a class with the same name. This utility provides an API for L2 Robot Keywords to control `kubectl` installed on each of DUTs. One time initialization script, `resources/libraries/bash/k8s_setup.sh` does reset/init kubectl, and initializes the `csit` namespace. CSIT namespace is required to not to interfere with existing setups and it further simplifies apply/get/delete Pod/ConfigMap operations on SUTs.

Kubernetes utility is based on YAML templates to avoid crafting the huge YAML configuration files, what would lower the readability of code and requires complicated algorithms.

Two types of YAML templates are defined:

- Static - do not change between deployments, that is infrastructure containers like Kafka, Calico, ETCD.
- Dynamic - per test suite/case topology YAML files.

Making own python wrapper library of `kubectl` instead of using the official Python package allows to control and deploy environment over the SSH library without the need of using isolated driver running on each of DUTs.

Tested Topologies

Listed CSIT container networking test topologies are defined with DUT containerized VPP switch forwarding packets between NF containers. Each NF container runs their own instance of VPP in L2XC configuration.

Following container networking topologies are tested in CSIT-1908.2:

- LXC topologies:
  - eth-l2xcbase-eth-2memif-1lxc.
  - eth-l2bdbasemaclrn-eth-2memif-1lxc.
- Docker topologies:
  - eth-l2xcbase-eth-2memif-1docker.
  - eth-l2xcbase-eth-1memif-1docker

References

2.7.2 Test Code Documentation

CSIT VPP Performance Tests Documentation[91] contains detailed functional description and input parameters for each test case.

[91] https://docs.fd.io/csit/rls1908_2/doc/tests.vpp.perf.html

CHAPTER

THREE

VPP DEVICE

3.1 Overview

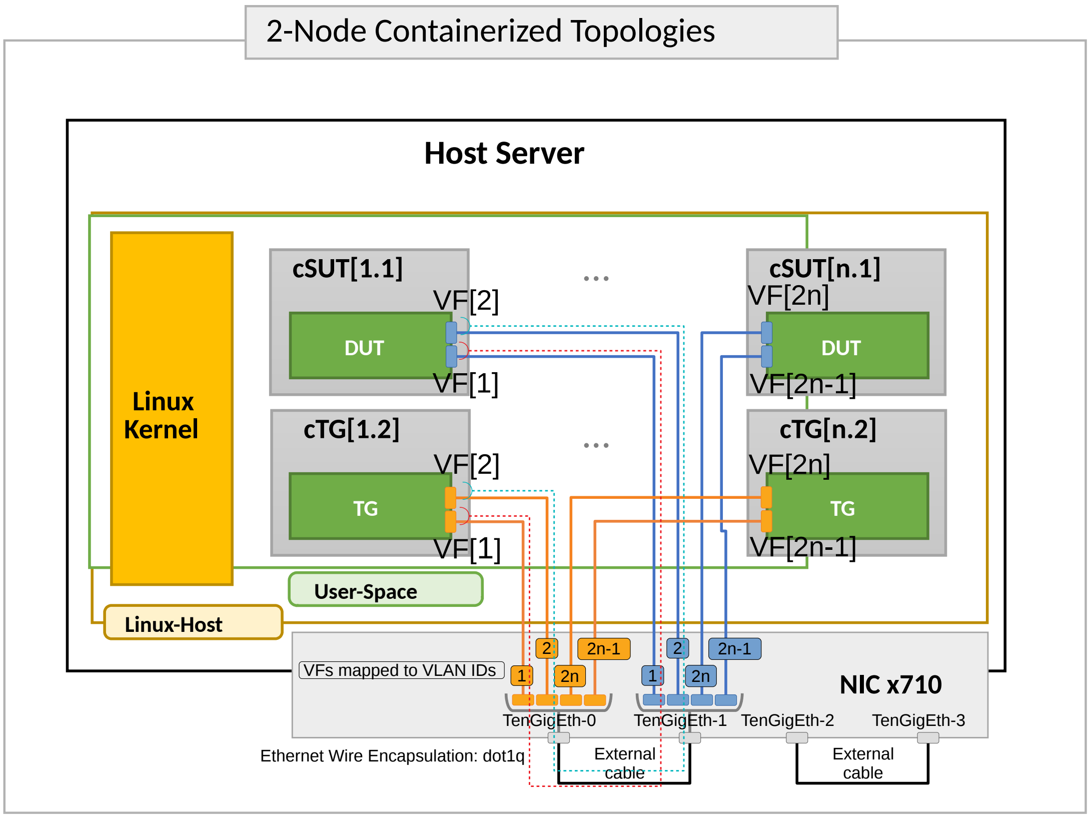
3.1.1 Virtual Topologies

CSIT VPP Device tests are executed in Physical containerized topologies created on demand using set of scripts hosted and developed under CSIT repository. It runs on physical baremetal servers hosted by LF FD.io project. Based on the packet path thru SUT Containers, three distinct logical topology types are used for VPP DUT data plane testing:

1. vfNIC-to-vfNIC switching topologies.
2. vfNIC-to-vhost-user switching topologies.
3. vfNIC-to-memif switching topologies.

vfNIC-to-vfNIC Switching

The simplest physical topology for software data plane application like VPP is vfNIC-to-vfNIC switching. Tested virtual topologies for 2-Node testbeds are shown in figures below.

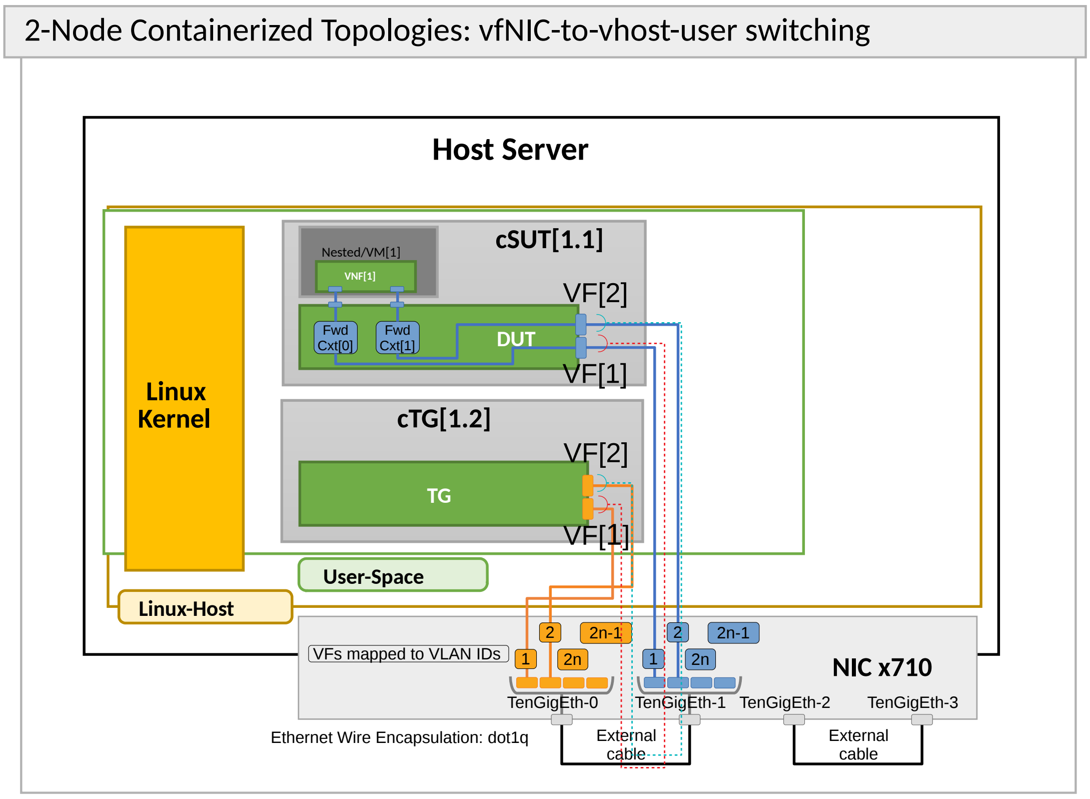


SUT1 is Docker Container (running Ubuntu, depending on the test suite), TG is a Traffic Generator (running Ubuntu Container). SUTs run VPP SW application in Linux user-mode as a Device Under Test (DUT) within the container. TG runs Scapy SW application as a packet Traffic Generator. Network connectivity between SUTs and to TG is provided using virtual function of physical NICs.

Virtual topologies are created on-demand whenever a verification job is started (e.g. triggered by the gerrit patch submission) and destroyed upon completion of all functional tests. Each node is a container running on physical server. During the test execution, all nodes are reachable thru the Management (not shown above for clarity).

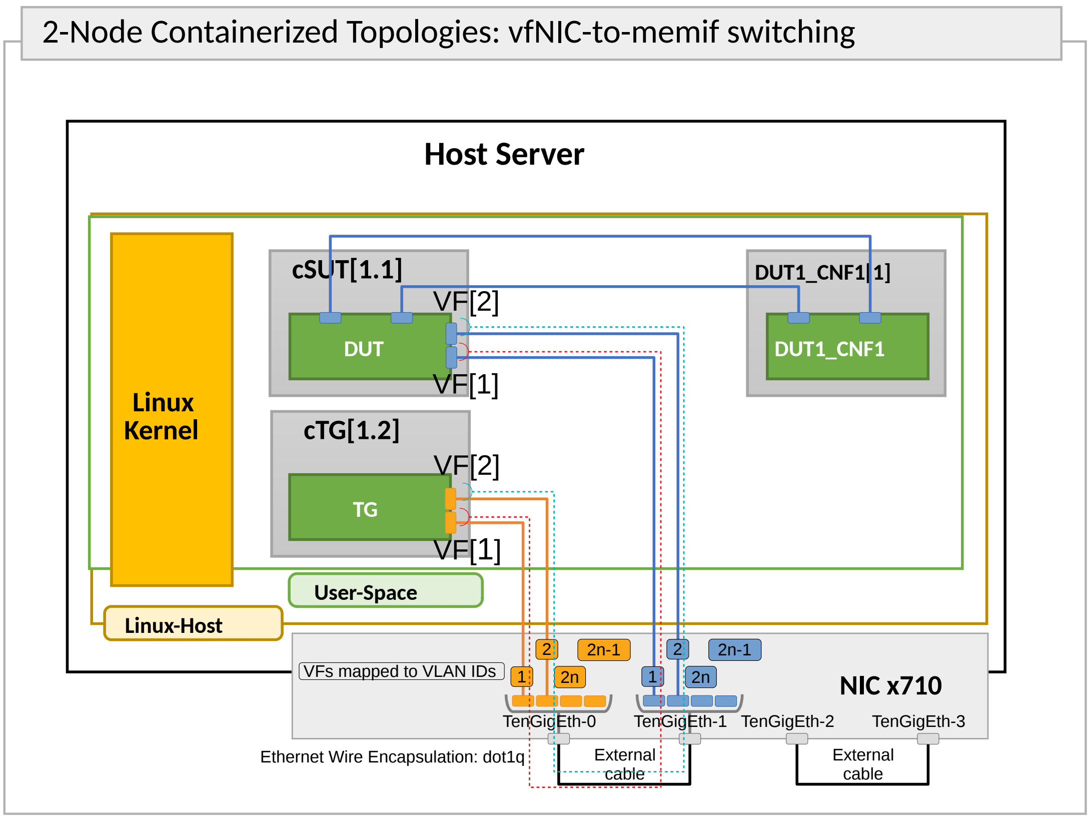
vfNIC-to-vhost-user Switching

vfNIC-to-vhost-user switching topology test cases require VPP DUT to communicate with Virtual Machine (VM) over Vhost-user virtual interfaces. VM is created on SUT1 for the duration of these particular test cases only. Virtual test topology with VM is shown in the figure below.



vfNIC-to-memif Switching

vfNIC-to-memif switching topology test cases require VPP DUT to communicate with another Docker Container over memif interfaces. Container is created for the duration of these particular test cases only and it is running the same VPP version as running on DUT. Virtual test topology with Memif is shown in the figure below.



3.1.2 Functional Tests Coverage

CSIT-1908.2 includes following VPP functionality tested in VPP Device environment:

Functionality	Description
ACL	Ingress Access Control List security for L2 Bridge-Domain MAC switching, IPv4 routing, IPv6 routing.
COP	COP address white-list and black-list filtering for IPv4 and IPv6 routing.
IPSec	IPSec tunnel and transport modes.
IPv4	IPv4 routing, ICMPv4.
IPv6	IPv4 routing, ICMPv6.
L2BD	L2 Bridge-Domain switching for untagged Ethernet.
L2XC	L2 Cross-Connect switching for untagged Ethernet.
Memif Interface	Baseline VPP memif interface tests.
QoS Policer Metering	Ingress packet rate metering and marking for IPv4, IPv6.
Tap Interface	Baseline Linux tap interface tests.
VLAN Tag	L2 VLAN subinterfaces.
Vhost-user Interface	Baseline VPP vhost-user interface tests.
VXLAN	VXLAN overlay tunneling for L2-over-IPv4 and -over-IPv6.

3.1.3 Tests Naming

CSIT-1908.2 follows a common structured naming convention for all performance and system functional tests, introduced in CSIT-17.01.

The naming should be intuitive for majority of the tests. Complete description of CSIT test naming convention is provided on *Test Naming* (page 185).

3.2 Release Notes

3.2.1 Changes in CSIT-1908.2

1. TEST FRAMEWORK
   - **Bug fixes**.
   - **ARM platform compatibility**.
2. TEST COVERAGE
   - Increased test coverage: **Dot1q**, **IPsec**, **802.1ad VXLAN**, **COP whitelist**, **COP blacklist**, **QoS Policer Metering**, **iACL whitelist**, **AVF driver**, **TAP Interface**.
   - Align vpp_device L2 Robot Keywords with performance L2 Robot Keywords.

3.2.2 Known Issues

List of known issues in CSIT-1908.2 for VPP functional tests in VPP Device:

#	JiraID	Issue Description
1		

3.3 Integration Tests

3.3.1 Abstract

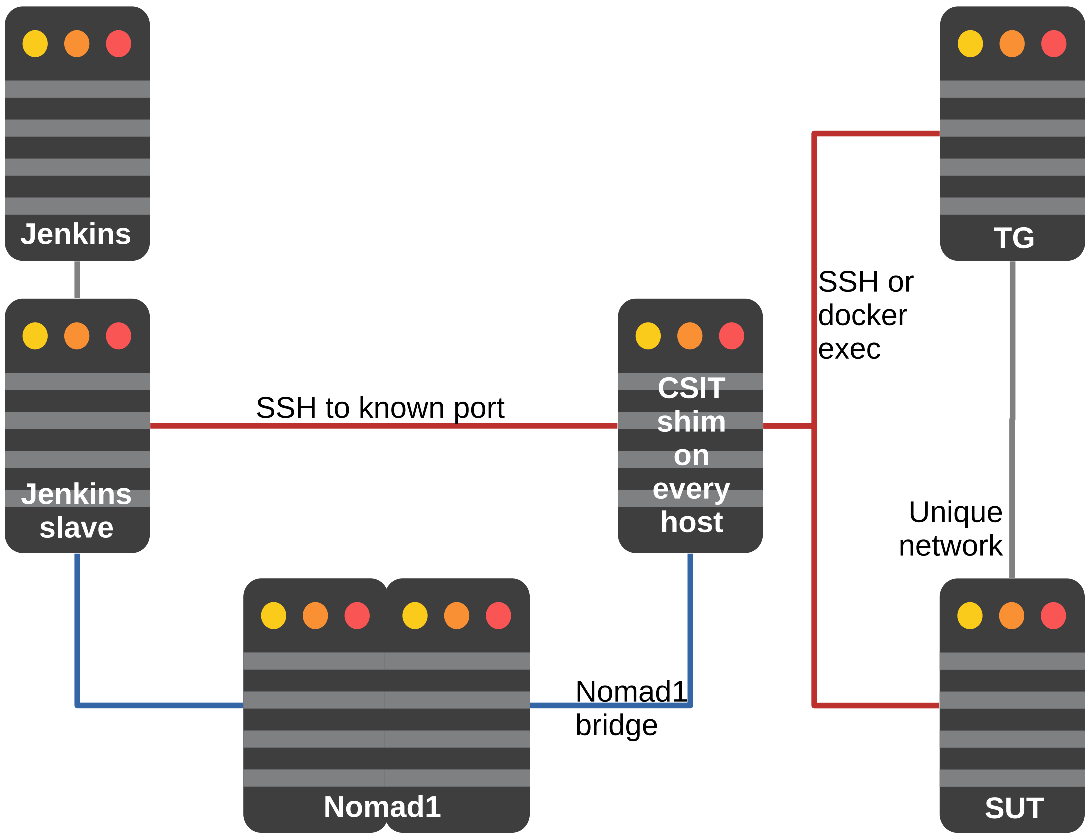
FD.io VPP software data plane technology has become very popular across a wide range of VPP ecosystem use cases, putting higher pressure on continuous verification of VPP software quality.

This document describes a proposal for design and implementation of extended continuous VPP testing by extending existing test environments. Furthermore it describes and summarizes implementation details of Integration and System tests platform *1-Node VPP_Device*. It aims to provide a complete end-to-end view of *1-Node VPP_Device* environment in order to improve extendability and maintenance, under the guideline of VPP core team.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 8174[92].

[92] https://tools.ietf.org/html/rfc8174.html

3.3.2 Overview



3.3.3 Physical Testbeds

All FD.io CSIT vpp-device tests are executed on physical testbeds built with bare-metal servers hosted by LF FD.io project. Two 1-node testbed topologies are used:

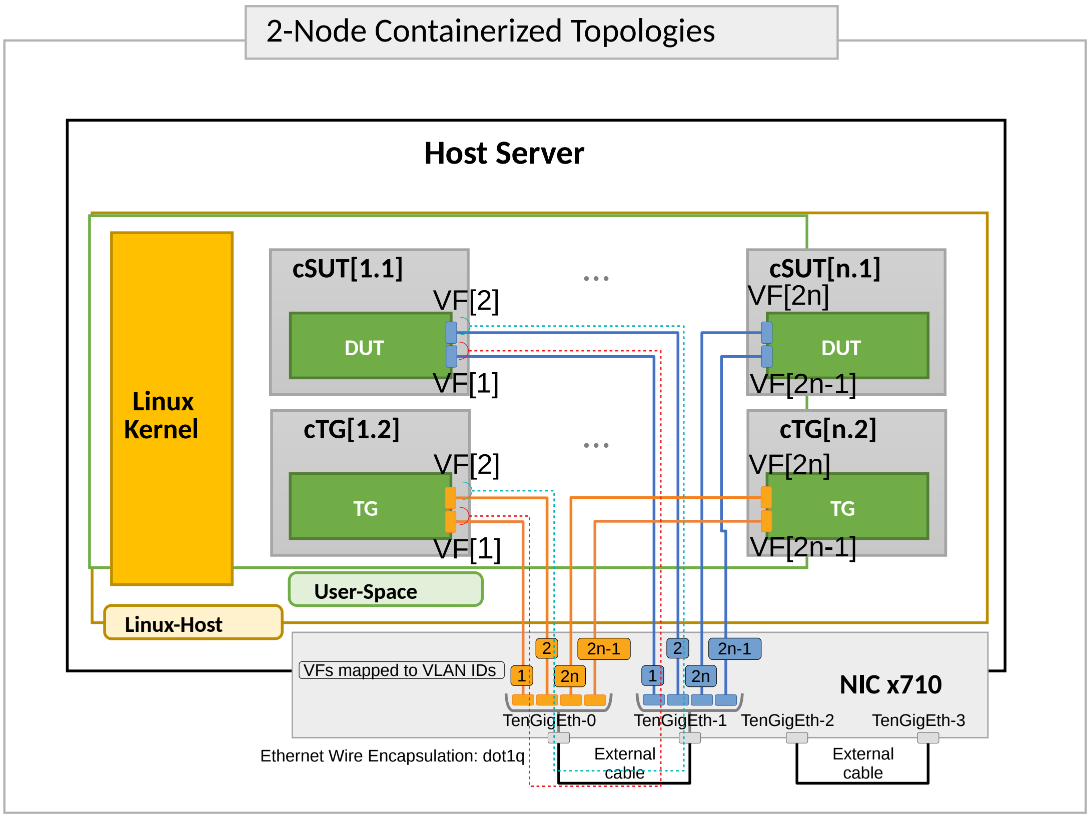
- **2-Container Topology**: Consisting of one Docker container acting as SUT (System Under Test) and one Docker container as TG (Traffic Generator), both connected in ring topology via physical NIC cross-connecting.

Current FD.io production testbeds are built with servers based on one processor generation of Intel Xeons: Skylake (Platinum 8180). Testbeds built with servers based on Arm processors are in the process of being added to FD.io production.

Following section describe existing production 1n-skx testbed.

1-Node Xeon Skylake (1n-skx)

1n-skx testbed is based on single SuperMicro SYS-7049GP-TRT server equipped with two Intel Xeon Skylake Platinum 8180 2.5 GHz 28 core processors. Physical testbed topology is depicted in a figure below.



Server is populated with the following NIC models:

1. NIC-1: x710-da4 4p10GE Intel.
2. NIC-2: x710-da4 4p10GE Intel.

All Intel Xeon Skylake servers run with Intel Hyper-Threading enabled, doubling the number of logical cores exposed to Linux, with 56 logical cores and 28 physical cores per processor socket.

NIC interfaces are shared using Linux vfio_pci and VPP VF drivers:

- DPDK VF driver,
- Fortville AVF driver.

Provided Intel x710-da4 4p10GE NICs support 32 VFs per interface, 128 per NIC.

Complete 1n-skx testbeds specification is available on CSIT LF Testbeds[93] wiki page.

Total of two 1n-skx testbeds are in operation in FD.io labs.

1-Node Virtualbox (1n-vbox)

1n-skx testbed can run in single VirtualBox VM machine. This solution replaces the previously used Vagrant environment based on 3 VMs.

VirtualBox VM MAY be created by Vagrant and MUST have additional 4 virtio NICs each pair attached to separate private networks to simulate back-to-back connections. It SHOULD be 82545EM device model (otherwise can be changed in boostrap scripts). Example of Vagrant configuration:

```
Vagrant.configure(2) do |c|
  c.vm.network "private_network", type: "dhcp", auto_config: false,
```

(continues on next page)

[93] https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.

(continued from previous page)

```
      virtualbox__intnet: "port1", nic_type: "82545EM"
  c.vm.network "private_network", type: "dhcp", auto_config: false,
      virtualbox__intnet: "port2", nic_type: "82545EM"

  c.vm.provider :virtualbox do |v|
    v.customize ["modifyvm", :id, "--nicpromisc2", "allow-all"]
    v.customize ["modifyvm", :id, "--nicpromisc3", "allow-all"]
    v.customize ["modifyvm", :id, "--nicpromisc4", "allow-all"]
    v.customize ["modifyvm", :id, "--nicpromisc5", "allow-all"]
```

Vagrant VM is populated with the following NIC models:

1. NIC-1: 82545EM Intel.
2. NIC-2: 82545EM Intel.
3. NIC-3: 82545EM Intel.
4. NIC-4: 82545EM Intel.

3.3.4 Containers

It was agreed on TWS (Technical Work Stream) call to continue with Ubuntu 18.04 LTS as a baseline system with OPTIONAL extend to Centos 7 and SuSE per demand [TWSLink].

All DCR (Docker container) images are REQUIRED to be hosted on Docker registry available from LF network, publicly available and trackable. For backup, tracking and contributing purposes all Dockerfiles (including files needed for building container) MUST be available and stored in [fdiocsitgerrit] repository under appropriate folders. This allows the peer review process to be done for every change of infrastructure related to scope of this document. Currently only **csit-shim-dcr** and **csit-sut-dcr** containers will be stored and maintained under CSIT repository by CSIT contributors.

At the time of designing solution described in this document the interconnection between [dockerhub] and [fdiocsitgerrit] for automated build purposes and image hosting cannot be established with the trust and respectful to security of FD.io project. Unless adressed, DCR images will be placed in custom registry service [fdioregistry]. Automated Jenkins jobs will be created in align of long term solution for container lifecycle and ability to build new version of docker images.

In parallel, the effort is started to find the outsourced Docker registry service.

Versioning

As of initial version of vpp-device, we do have only single latest version of Docker image hosted on [dockerhub]. This will be addressed as further improvement with proper semantic versioning.

jenkins-slave-dcr

This DCR acts as the Jenkins slave (known also as jenkins minion). It can connect over SSH protocol to TCP port 6022 of **csit-shim-dcr** and executes non-interactive reservation script. Nomad is responsible for scheduling this container execution onto specific **1-Node VPP_Device** testbed. It executes CSIT environment including CSIT framework.

All software dependencies including VPP/DPDK that are not present in **csit-sut-dcr** container image and/or needs to be compiled prior running on **csit-sut-dcr** SHOULD be compiled in this container.

- *Container Image Location*: Docker image at snergster/vpp-ubuntu18.
- *Container Definition*: Docker file specified at [JenkinsSlaveDcrFile].
- *Initializing*: Container is initialized from within *Consul by HashiCorp* and *Nomad by HashiCorp*.

csit-shim-dcr

This DCR acts as an intermediate layer running script responsible for orchestrating topologies under test and reservation. Responsible for managing VF resources and allocation to DUT (Device Under Test), TG (Traffic Generator) containers. This MUST to be done on **csit-shim-dcr**. This image also acts as the generic reservation mechanics arbiter to make sure that only Y number of simulations are spawned on any given HW node.

- *Container Image Location*: Docker image at snergster/csit-shim.
- *Container Definition*: Docker file specified at [CsitShimDcrFile].
- *Initializing*: Container is initialized from within *Consul by HashiCorp* and *Nomad by HashiCorp*. Required docker parameters, to be able to run nested containers with VF reservation system are: privileged, net=host, pid=host.
- *Connectivity*: Over SSH only, using <host>:6022 format. Currently using *root* user account as primary. From the jenkins slave it will be able to connect via env variable, since the jenkins slave doesn't actually know what host its running on.

```
ssh -p 6022 root@10.30.51.node
```

csit-sut-dcr

This DCR acts as an SUT (System Under Test). Any DUT or TG application is installed there. It is RECOMMENDED to install DUT and all DUT dependencies via commands `rpm -ihv` on RedHat based OS or `dpkg -i` on Debian based OS.

Container is designed to be a very lightweight Docker image that only installs packages and execute binaries (previously built or downloaded on **jenkins-slave-dcr**) and contains libraries necessary to run CSIT framework including those required by DUT/TG.

- *Container Image Location*: Docker image at snergster/csit-sut.
- *Container Definition*: Docker file specified at [CsitSutDcrFile].
- *Initializing*:

```
docker run
# Run the container in the background and print the new container ID.
--detach=true
# Give extended privileges to this container. A "privileged" container is
# given access to all devices and able to run nested containers.
--privileged
# Publish all exposed ports to random ports on the host interfaces.
--publish-all
# Automatically remove the container when it exits.
--rm
# Size of /dev/shm.
dcr_stc_params+="--shm-size 512M "
# Override access to PCI bus by attaching a filesystem mount to the
# container.
dcr_stc_params+="--mount type=tmpfs,destination=/sys/bus/pci/devices "
# Mount vfio to be able to bind to see bound interfaces. We cannot use
# --device=/dev/vfio as this does not see newly bound interfaces.
dcr_stc_params+="--volume /dev/vfio:/dev/vfio "
# Mount docker.sock to be able to use docker deamon of the host.
dcr_stc_params+="--volume /var/run/docker.sock:/var/run/docker.sock "
# Mount /opt/boot/ where VM kernel and initrd are located.
dcr_stc_params+="--volume /opt/boot/:/opt/boot/ "
# Mount host hugepages for VMs.
dcr_stc_params+="--volume /dev/hugepages/:/dev/hugepages/ "
```

Container name is catenated from **csit-** prefix and uuid generated uniquely for each container instance.

- *Connectivity*: Over SSH only, using <host>[:<port>] format. Currently using *root* user account as primary.

```
ssh -p <port> root@10.30.51.<node>
```

Container required to run as `--privileged` due to ability to create nested containers and have full read/write access to sysfs (for bind/unbind). Docker automatically pick free network port (`--publish-all`) for ability to connect over ssh. To be able to limit access to PCI bus, container is creating tmpfs mount type in PCI bus tree. CSIT reservation script is dynamically linking only PCI devices (NIC cards) that are reserved for particular container. This way it is not colliding with other containers. To make vfio work, access to `/dev/vfio` must be granted.

3.3.5 Environment initialization

All 1-node servers are to be managed and provisioned via the [ansiblelink] set of playbooks with *vpp-device* role. Full playbooks can be found under [fdiocsitansible] directory. This way we are able to track all configuration changes of physical servers in gerrit (in structured yaml format) as well as we are able to extend *vpp-device* to additional servers with less effort or re-stage servers in case of failure.

SR-IOV VF initialization is done via `systemd` service during host system boot up. Service with name *csit-initialize-vfs.service* is created under systemd system context (`/etc/systemd/system/`). By default service is calling `/usr/local/bin/csit-initialize-vfs.sh` with single parameter:

- **start**: Creates maximum number of virtual functions (VFs) (detected from `sriov_totalvfs`) for each whitelisted PCI device.
- **stop**: Removes all VFs (VFs) for all whitelisted PCI device.

Service is considered active even when all of its processes exited successfully. Stopping service will automatically remove VFs.

```
[Unit]
Description=CSIT Initialize SR-IOV VFs
After=network.target

[Service]
Type=one-shot
RemainAfterExit=True
ExecStart=/usr/local/bin/csit-initialize-vfs.sh start
ExecStop=/usr/local/bin/csit-initialize-vfs.sh stop

[Install]
WantedBy=default.target
```

Script is driven by two array variables `pci_blacklist/pci_whitelist`. They MUST store all PCI addresses in **<domain>:<bus>:<device>.<func>** format, where:

- **pci_blacklist**: PCI addresses to be skipped from VFs initialization (usefull for e.g. excluding management network interfaces).
- **pci_whitelist**: PCI addresses to be included for VFs initialization.

3.3.6 VF reservation

During topology initialization phase of script, mutex is used to avoid multiple instances of script to interact with each other during resources allocation. Mutal exclusion ensure that no two distinct instances of script will get same resource list.

Reservation function reads the list of all available virtual function network devices in system:

```
# Find the first ${device_count} number of available TG Linux network
# VF device names. Only allowed VF PCI IDs are filtered.
for netdev in ${tg_netdev[@]}
do
    for netdev_path in $(grep -l "${pci_id}" \
                         /sys/class/net/${netdev}*/device/device \
                         2> /dev/null)
    do
        if [[ ${#TG_NETDEVS[@]} -lt ${device_count} ]]; then
            tg_netdev_name=$(dirname ${netdev_path})
            tg_netdev_name=$(dirname ${tg_netdev_name})
            TG_NETDEVS+=($(basename ${tg_netdev_name}))
        else
            break
        fi
    done
    if [[ ${#TG_NETDEVS[@]} -eq ${device_count} ]]; then
        break
    fi
done
```

Where `${pci_id}` is ID of white-listed VF PCI ID. For more information please see [pciids]. This act as security constraint to prevent taking other unwanted interfaces. The output list of all VF network devices is split into two lists for TG and SUT side of connection. First two items from each TG or SUT network devices list are taken to expose directly to namespace of container. This can be done via commands:

```
$ ip link set ${netdev} netns ${DCR_CPIDS[tg]}
$ ip link set ${netdev} netns ${DCR_CPIDS[dut1]}
```

In this stage also symbolic links to PCI devices under sysfs bus directory tree are created in running containers. Once VF devices are assigned to container namespace and PCI deivces are linked to running containers and mutex is exited. Selected VF network device automatically dissapear from parent container namespace, so another instance of script will not find device under that namespace.

Once Docker container exits, network device is returned back into parent namespace and can be reused.

3.3.7 Network traffic isolation - Intel i40evf

In a virtualized environment, on Intel(R) Server Adapters that support SR-IOV, the virtual function (VF) may be subject to malicious behavior. Software- generated layer two frames, like IEEE 802.3x (link flow control), IEEE 802.1Qbb (priority based flow-control), and others of this type, are not expected and can throttle traffic between the host and the virtual switch, reducing performance. To resolve this issue, configure all SR-IOV enabled ports for VLAN tagging. This configuration allows unexpected, and potentially malicious, frames to be dropped. [inteli40e]

To configure VLAN tagging for the ports on an SR-IOV enabled adapter, use the following command. The VLAN configuration SHOULD be done before the VF driver is loaded or the VM is booted. [inteli40e]

```
$ ip link set dev <PF netdev id> vf <id> vlan <vlan id>
```

For example, the following instructions will configure PF eth0 and the first VF on VLAN 10.

```
$ ip link set dev eth0 vf 0 vlan 10
```

VLAN Tag Packet Steering allows to send all packets with a specific VLAN tag to a particular SR-IOV virtual function (VF). Further, this feature allows to designate a particular VF as trusted, and allows that trusted VF to request selective promiscuous mode on the Physical Function (PF). [inteli40e]

To set a VF as trusted or untrusted, enter the following command in the Hypervisor:

```
$ ip link set dev eth0 vf 1 trust [on|off]
```

Once the VF is designated as trusted, use the following commands in the VM to set the VF to promiscuous mode. [inteli40e]

- For promiscuous all:

  ```
  $ ip link set eth2 promisc on
  ```

- For promiscuous Multicast:

  ```
  $ ip link set eth2 allmulti on
  ```

Note: By default, the ethtool priv-flag vf-true-promisc-support is set to *off*, meaning that promiscuous mode for the VF will be limited. To set the promiscuous mode for the VF to true promiscuous and allow the VF to see all ingress traffic, use the following command. $ ethtool set-priv-flags p261p1 vf-true-promisc-support on The vf-true-promisc-support priv-flag does not enable promiscuous mode; rather, it designates which type of promiscuous mode (limited or true) you will get when you enable promiscuous mode using the ip link commands above. Note that this is a global setting that affects the entire device. However,the vf-true-promisc-support priv-flag is only exposed to the first PF of the device. The PF remains in limited promiscuous mode (unless it is in MFP mode) regardless of the vf-true-promisc-support setting. [inteli40e]

Service described earlier *csit-initialize-vfs.service* is responsible for assigning 802.1Q vlan tagging to each vitual function via physical function from list of white-listed PCI addresses by following (simplified) code.

```
SCRIPT_DIR="$(dirname $(readlink -e "${BASH_SOURCE[0]}"))"
source "${SCRIPT_DIR}/csit-initialize-vfs-data.sh"

# Initilize whitelisted NICs with maximum number of VFs.
pci_idx=0
for pci_addr in ${PCI_WHITELIST[@]}; do
    if ! [[ ${PCI_BLACKLIST[*]} =~ "${pci_addr}" ]]; then
        pci_path="/sys/bus/pci/devices/${pci_addr}"
        # SR-IOV initialization
        case "${1:-start}" in
            "start" )
                sriov_totalvfs=$(< "${pci_path}"/sriov_totalvfs)
                ;;
            "stop" )
                sriov_totalvfs=0
                ;;
        esac
        echo ${sriov_totalvfs} > "${pci_path}"/sriov_numvfs
        # SR-IOV 802.1Q isolation
        case "${1:-start}" in
            "start" )
                pf=$(basename "${pci_path}"/net/*)
                for vf in $(seq "${sriov_totalvfs}"); do
                    # PCI address index in array (pairing siblings).
                    if [[ -n ${PF_INDICES[@]} ]]
                    then
                        vlan_pf_idx=${PF_INDICES[$pci_addr]}
                    else
                        vlan_pf_idx=$((pci_idx % (${#PCI_WHITELIST[@]}/2)))
                    fi
                    # 802.1Q base offset.
                    vlan_bs_off=1100
                    # 802.1Q PF PCI address offset.
```

(continues on next page)

(continued from previous page)

```
                    vlan_pf_off=$(( vlan_pf_idx * 100 + vlan_bs_off ))
                    # 802.1Q VF PCI address offset.
                    vlan_vf_off=$(( vlan_pf_off + vf - 1 ))
                    # VLAN string.
                    vlan_str="vlan ${vlan_vf_off}"
                    # MAC string.
                    mac5="$(printf '%x' ${pci_idx})"
                    mac6="$(printf '%x' $(( vf - 1 )))"
                    mac_str="mac ba:dc:0f:fe:${mac5}:${mac6}"
                    # Set 802.1Q VLAN id and MAC address
                    ip link set ${pf} vf $(( vf - 1)) ${mac_str} ${vlan_str}
                    ip link set ${pf} vf $(( vf - 1)) trust on
                    ip link set ${pf} vf $(( vf - 1)) spoof off
                done
                pci_idx=$(( pci_idx + 1 ))
                ;;
        esac
        rmmod i40evf
        modprobe i40evf
    fi
done
```

Assignment starts at VLAN 1100 and incrementing by 1 for each VF and by 100 for each white-listed PCI address up to the middle of the PCI list. Second half of the lists is assumed to be directly (cable) paired siblings and assigned with same 802.1Q VLANs as its siblings.

3.3.8 Open tasks

Security

Note: Switch to non-privileged containers: As of now all three container flavors are using privileged containers to make it working. Explore options to switch containers to non-privileged with explicit rather implicit privileges.

Note: Switch to testuser account intead of root.

Maintainability

Note: Docker image distribution: Create jenkins jobs with full pipiline of CI/CD for CSIT Docker images.

Stability

Note: Implement queueing mechanism: Currently there is no mechanics that would place starving jobs in queue in case of no resources available.

Note: Replace reservation script with Docker network plugin written in GOLANG/SH/Python - platform independent.

3.3.9 Links

3.4 Documentation

CSIT VPP Device Tests Documentation[104] contains detailed functional description and input parameters for each test case.

[104] https://docs.fd.io/csit/rls1908_2/doc/tests.vpp.device.html

CHAPTER

FOUR

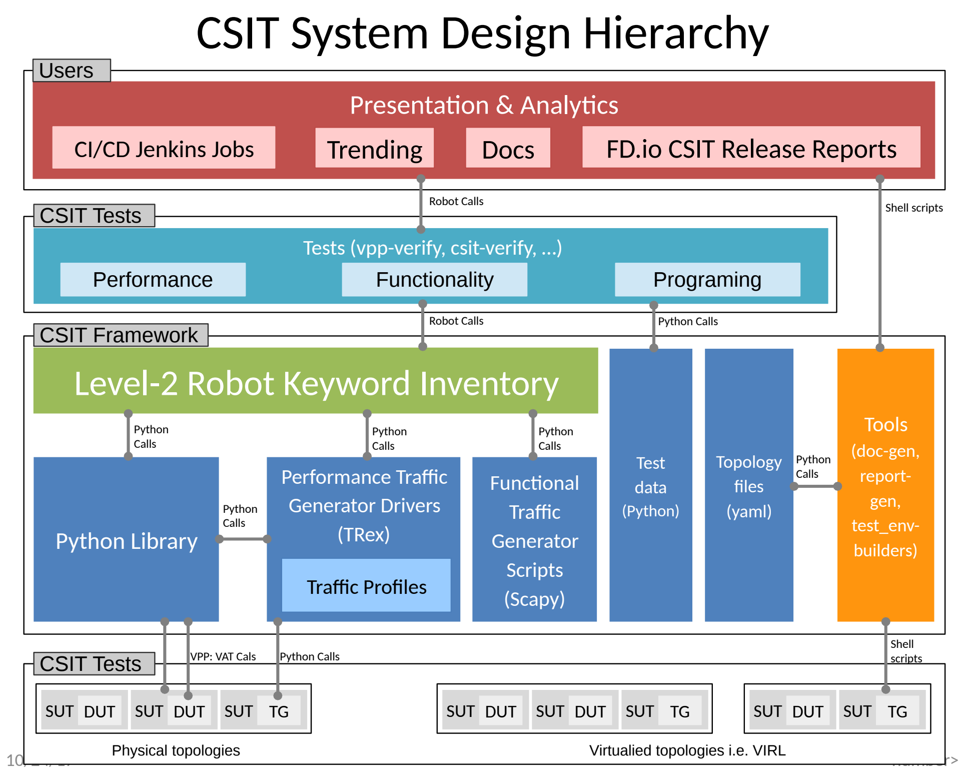
CSIT FRAMEWORK

4.1 Design

FD.io CSIT system design needs to meet continuously expanding requirements of FD.io projects including VPP, related sub-systems (e.g. plugin applications, DPDK drivers) and FD.io applications (e.g. DPDK applications), as well as growing number of compute platforms running those applications. With CSIT project scope and charter including both FD.io continuous testing AND performance trending/comparisons, those evolving requirements further amplify the need for CSIT framework modularity, flexibility and usability.

4.1.1 Design Hierarchy

CSIT follows a hierarchical system design with SUTs and DUTs at the bottom level of the hierarchy, presentation level at the top level and a number of functional layers in-between. The current CSIT system design including CSIT framework is depicted in the figure below.



A brief bottom-up description is provided here:

1. SUTs, DUTs, TGs
   - SUTs - Systems Under Test;
   - DUTs - Devices Under Test;
   - TGs - Traffic Generators;
2. Level-1 libraries - Robot and Python
   - Lowest level CSIT libraries abstracting underlying test environment, SUT, DUT and TG specifics;
   - Used commonly across multiple L2 KWs;
   - Performance and functional tests:
     - L1 KWs (KeyWords) are implemented as RF libraries and Python libraries;
   - Performance TG L1 KWs:
     - All L1 KWs are implemented as Python libraries:
       * Support for TRex only today;
       * CSIT IXIA drivers in progress;
   - Performance data plane traffic profiles:
     - TG-specific stream profiles provide full control of:
       * Packet definition - layers, MACs, IPs, ports, combinations thereof e.g. IPs and UDP ports;
       * Stream definitions - different streams can run together, delayed, one after each other;
       * Stream profiles are independent of CSIT framework and can be used in any T-rex setup, can be sent anywhere to repeat tests with exactly the same setup;
       * Easily extensible - one can create a new stream profile that meets tests requirements;
       * Same stream profile can be used for different tests with the same traffic needs;
   - Functional data plane traffic scripts:
     - Scapy specific traffic scripts;
3. Level-2 libraries - Robot resource files:
   - Higher level CSIT libraries abstracting required functions for executing tests;
   - L2 KWs are classified into the following functional categories:
     - Configuration, test, verification, state report;
     - Suite setup, suite teardown;
     - Test setup, test teardown;
4. Tests - Robot:
   - Test suites with test cases;
   - Performance tests using physical testbed environment:
     - VPP;
     - DPDK-Testpmd;
     - DPDK-L3Fwd;
   - Tools:
     - Documentation generator;
     - Report generator;

- Testbed environment setup ansible playbooks;
- Operational debugging scripts;

4.1.2 Test Lifecycle Abstraction

A well coded test must follow a disciplined abstraction of the test lifecycles that includes setup, configuration, test and verification. In addition to improve test execution efficiency, the commmon aspects of test setup and configuration shared across multiple test cases should be done only once. Translating these high-level guidelines into the Robot Framework one arrives to definition of a well coded RF tests for FD.io CSIT. Anatomy of Good Tests for CSIT:

1. Suite Setup - Suite startup Configuration common to all Test Cases in suite: uses Configuration KWs, Verification KWs, StateReport KWs;
2. Test Setup - Test startup Configuration common to multiple Test Cases: uses Configuration KWs, StateReport KWs;
3. Test Case - uses L2 KWs with RF Gherkin style:
   - prefixed with {Given} - Verification of Test setup, reading state: uses Configuration KWs, Verification KWs, StateReport KWs;
   - prefixed with {When} - Test execution: Configuration KWs, Test KWs;
   - prefixed with {Then} - Verification of Test execution, reading state: uses Verification KWs, StateReport KWs;
4. Test Teardown - post Test teardown with Configuration cleanup and Verification common to multiple Test Cases - uses: Configuration KWs, Verification KWs, StateReport KWs;
5. Suite Teardown - Suite post-test Configuration cleanup: uses Configuration KWs, Verification KWs, StateReport KWs;

4.1.3 RF Keywords Functional Classification

CSIT RF KWs are classified into the functional categories matching the test lifecycle events described earlier. All CSIT RF L2 and L1 KWs have been grouped into the following functional categories:

1. Configuration;
2. Test;
3. Verification;
4. StateReport;
5. SuiteSetup;
6. TestSetup;
7. SuiteTeardown;
8. TestTeardown;

4.1.4 RF Keywords Naming Guidelines

Readability counts: "..code is read much more often than it is written." Hence following a good and consistent grammar practice is important when writing RF KeyWords and Tests. All CSIT test cases are coded using Gherkin style and include only L2 KWs references. L2 KWs are coded using simple style and include L2 KWs, L1 KWs, and L1 python references. To improve readability, the proposal is to use the same grammar for both RF KW styles, and to formalize the grammar of English sentences used for naming the RF KWs. RF KWs names are short sentences expressing functional description of the command. They must follow English sentence grammar in one of the following forms:

1. **Imperative** - verb-object(s): *"Do something"*, verb in base form.
2. **Declarative** - subject-verb-object(s): *"Subject does something"*, verb in a third-person singular present tense form.
3. **Affirmative** - modal_verb-verb-object(s): *"Subject should be something"*, *"Object should exist"*, verb in base form.
4. **Negative** - modal_verb-Not-verb-object(s): *"Subject should not be something"*, *"Object should not exist"*, verb in base form.

Passive form MUST NOT be used. However a usage of past participle as an adjective is okay. See usage examples provided in the Coding guidelines section below. Following sections list applicability of the above grammar forms to different RF KW categories. Usage examples are provided, both good and bad.

4.1.5 Coding Guidelines

Coding guidelines can be found on Design optimizations wiki page[105].

4.2 Test Naming

4.2.1 Background

CSIT-1908.2 follows a common structured naming convention for all performance and system functional tests, introduced in CSIT-1701.

The naming should be intuitive for majority of the tests. Complete description of CSIT test naming convention is provided on CSIT test naming wiki page[106]. Below few illustrative examples of the naming usage for test suites across CSIT performance, functional and Honeycomb management test areas.

4.2.2 Naming Convention

The CSIT approach is to use tree naming convention and to encode following testing information into test suite and test case names:

1. packet network port configuration
   - port type, physical or virtual;
   - number of ports;
   - NIC model, if applicable;
   - port-NIC locality, if applicable;
2. packet encapsulations;
3. VPP packet processing
   - packet forwarding mode;
   - packet processing function(s);
4. packet forwarding path
   - if present, network functions (processes, containers, VMs) and their topology within the computer;
5. main measured variable, type of test.

[105] https://wiki.fd.io/view/CSIT/Design_Optimizations
[106] https://wiki.fd.io/view/CSIT/csit-test-naming

Proposed convention is to encode ports and NICs on the left (underlay), followed by outer-most frame header, then other stacked headers up to the header processed by vSwitch-VPP, then VPP forwarding function, then encap on vhost interface, number of vhost interfaces, number of VMs. If chained VMs present, they get added on the right. Test topology is expected to be symmetric, in other words packets enter and leave SUT through ports specified on the left of the test name. Here some examples to illustrate the convention followed by the complete legend, and tables mapping the new test filenames to old ones.

4.2.3 Naming Examples

CSIT test suite naming examples (filename.robot) for common tested VPP topologies:

1. **Physical port to physical port - a.k.a. NIC-to-NIC, Phy-to-Phy, P2P**
   - *PortNICConfig-WireEncapsulation-PacketForwardingFunction- PacketProcessingFunction1-…-PacketProcessingFunctionN-TestType*
   - *10ge2p1x520-dot1q-l2bdbasemaclrn-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, dot1q tagged Ethernet, L2 bridge-domain baseline switching with MAC learning, NDR throughput discovery.
   - *10ge2p1x520-ethip4vxlan-l2bdbasemaclrn-ndrchk.robot* => 2 ports of 10GE on Intel x520 NIC, IPv4 VXLAN Ethernet, L2 bridge-domain baseline switching with MAC learning, NDR throughput discovery.
   - *10ge2p1x520-ethip4-ip4base-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, IPv4 baseline routed forwarding, NDR throughput discovery.
   - *10ge2p1x520-ethip6-ip6scale200k-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, IPv6 scaled up routed forwarding, NDR throughput discovery.
   - *10ge2p1x520-ethip4-ip4base-iacldstbase-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, IPv4 baseline routed forwarding, ingress Access Control Lists baseline matching on destination, NDR throughput discovery.
   - *40ge2p1vic1385-ethip4-ip4base-ndrdisc.robot* => 2 ports of 40GE on Cisco vic1385 NIC, IPv4 baseline routed forwarding, NDR throughput discovery.
   - *eth2p-ethip4-ip4base-func.robot* => 2 ports of Ethernet, IPv4 baseline routed forwarding, functional tests.
2. **Physical port to VM (or VM chain) to physical port - a.k.a. NIC2VM2NIC, P2V2P, NIC2VMchain2NIC, P2V2V2P**
   - *PortNICConfig-WireEncapsulation-PacketForwardingFunction- PacketProcessingFunction1-…-PacketProcessingFunctionN-VirtEncapsulation- VirtPortConfig-VMconfig-TestType*
   - *10ge2p1x520-dot1q-l2bdbasemaclrn-eth-2vhost-1vm-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, dot1q tagged Ethernet, L2 bridge-domain switching to/from two vhost interfaces and one VM, NDR throughput discovery.
   - *10ge2p1x520-ethip4vxlan-l2bdbasemaclrn-eth-2vhost-1vm-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, IPv4 VXLAN Ethernet, L2 bridge-domain switching to/from two vhost interfaces and one VM, NDR throughput discovery.
   - *10ge2p1x520-ethip4vxlan-l2bdbasemaclrn-eth-4vhost-2vm-ndrdisc.robot* => 2 ports of 10GE on Intel x520 NIC, IPv4 VXLAN Ethernet, L2 bridge-domain switching to/from four vhost interfaces and two VMs, NDR throughput discovery.
   - *eth2p-ethip4vxlan-l2bdbasemaclrn-eth-2vhost-1vm-func.robot* => 2 ports of Ethernet, IPv4 VXLAN Ethernet, L2 bridge-domain switching to/from two vhost interfaces and one VM, functional tests.
3. **API CRUD tests - Create (Write), Read (Retrieve), Update (Modify), Delete (Destroy) operations for configuration and operational data**

- *ManagementTestKeyword-ManagementOperation-ManagedFunction1-...- ManagedFunctionN-ManagementAPI1-ManagementAPIN-TestType*
- *mgmt-cfg-lisp-apivat-func* => configuration of LISP with VAT API calls, functional tests.
- *mgmt-cfg-l2bd-apihc-apivat-func* => configuration of L2 Bridge-Domain with Honeycomb API and VAT API calls, functional tests.
- *mgmt-oper-int-apihcnc-func* => reading status and operational data of interface with Honeycomb NetConf API calls, functional tests.
- *mgmt-cfg-int-tap-apihcnc-func* => configuration of tap interfaces with Honeycomb NetConf API calls, functional tests.
- *mgmt-notif-int-subint-apihcnc-func* => notifications of interface and sub-interface events with Honeycomb NetConf Notifications, functional tests.

For complete description of CSIT test naming convention please refer to CSIT test naming wiki page[107].

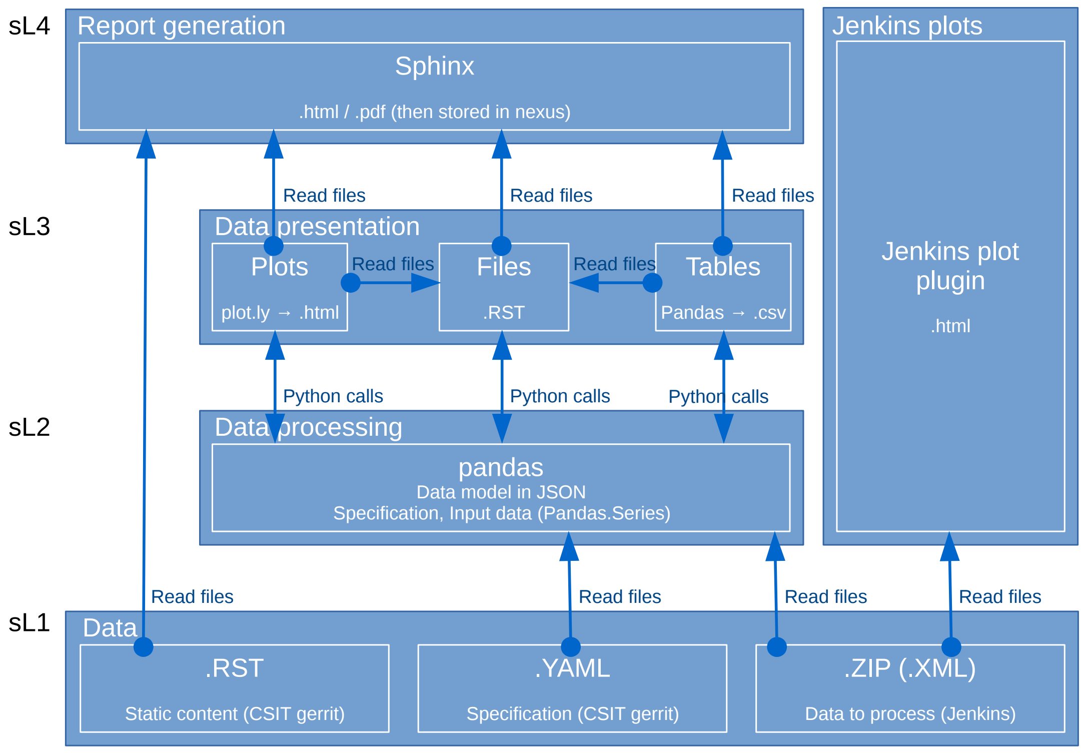
4.3 Presentation and Analytics

4.3.1 Overview

The presentation and analytics layer (PAL) is the fourth layer of CSIT hierarchy. The model of presentation and analytics layer consists of four sub-layers, bottom up:

- sL1 - Data - input data to be processed:
  - Static content - .rst text files, .svg static figures, and other files stored in the CSIT git repository.
  - Data to process - .xml files generated by Jenkins jobs executing tests, stored as robot results files (output.xml).
  - Specification - .yaml file with the models of report elements (tables, plots, layout, ...) generated by this tool. There is also the configuration of the tool and the specification of input data (jobs and builds).
- sL2 - Data processing
  - The data are read from the specified input files (.xml) and stored as multi-indexed pandas.Series[108].
  - This layer provides also interface to input data and filtering of the input data.
- sL3 - Data presentation - This layer generates the elements specified in the specification file:
  - Tables: .csv files linked to static .rst files.
  - Plots: .html files generated using plot.ly linked to static .rst files.
- sL4 - Report generation - Sphinx generates required formats and versions:
  - formats: html, pdf
  - versions: minimal, full (TODO: define the names and scope of versions)

[107] https://wiki.fd.io/view/CSIT/csit-test-naming
[108] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html



4.3.2 Data

Report Specification

The report specification file defines which data is used and which outputs are generated. It is human readable and structured. It is easy to add / remove / change items. The specification includes:

- Specification of the environment.
- Configuration of debug mode (optional).
- Specification of input data (jobs, builds, files, …).
- Specification of the output.
- What and how is generated: - What: plots, tables. - How: specification of all properties and parameters.
- .yaml format.

Structure of the specification file

The specification file is organized as a list of dictionaries distinguished by the type:

```
-
  type: "environment"
-
  type: "configuration"
-
  type: "debug"
-
  type: "static"
-
  type: "input"
-
```

(continues on next page)

(continued from previous page)

```
  type: "output"
-
  type: "table"
-
  type: "plot"
-
  type: "file"
```

Each type represents a section. The sections "environment", "debug", "static", "input" and "output" are listed only once in the specification; "table", "file" and "plot" can be there multiple times.

Sections "debug", "table", "file" and "plot" are optional.

Table(s), files(s) and plot(s) are referred as "elements" in this text. It is possible to define and implement other elements if needed.

Section: Environment

This section has the following parts:

- type: "environment" - says that this is the section "environment".
- configuration - configuration of the PAL.
- paths - paths used by the PAL.
- urls - urls pointing to the data sources.
- make-dirs - a list of the directories to be created by the PAL while preparing the environment.
- remove-dirs - a list of the directories to be removed while cleaning the environment.
- build-dirs - a list of the directories where the results are stored.

The structure of the section "Environment" is as follows (example):

```
-
  type: "environment"
  configuration:
    # Debug mode:
    # - Skip:
    #   - Download of input data files
    # - Do:
    #   - Read data from given zip / xml files
    #   - Set the configuration as it is done in normal mode
    # If the section "type: debug" is missing, CFG[DEBUG] is set to 0.
    CFG[DEBUG]: 0

  paths:
    # Top level directories:
    ## Working directory
    DIR[WORKING]: "_tmp"
    ## Build directories
    DIR[BUILD,HTML]: "_build"
    DIR[BUILD,LATEX]: "_build_latex"

    # Static .rst files
    DIR[RST]: "../../../docs/report"

    # Working directories
    ## Input data files (.zip, .xml)
    DIR[WORKING,DATA]: "{DIR[WORKING]}/data"
    ## Static source files from git
```

(continues on next page)

(continued from previous page)

```
    DIR[WORKING,SRC]: "{DIR[WORKING]}/src"
    DIR[WORKING,SRC,STATIC]: "{DIR[WORKING,SRC]}/_static"

    # Static html content
    DIR[STATIC]: "{DIR[BUILD,HTML]}/_static"
    DIR[STATIC,VPP]: "{DIR[STATIC]}/vpp"
    DIR[STATIC,DPDK]: "{DIR[STATIC]}/dpdk"
    DIR[STATIC,ARCH]: "{DIR[STATIC]}/archive"

    # Detailed test results
    DIR[DTR]: "{DIR[WORKING,SRC]}/detailed_test_results"
    DIR[DTR,PERF,DPDK]: "{DIR[DTR]}/dpdk_performance_results"
    DIR[DTR,PERF,VPP]: "{DIR[DTR]}/vpp_performance_results"
    DIR[DTR,FUNC,VPP]: "{DIR[DTR]}/vpp_functional_results"
    DIR[DTR,PERF,VPP,IMPRV]: "{DIR[WORKING,SRC]}/vpp_performance_tests/performance_improvements"

    # Detailed test configurations
    DIR[DTC]: "{DIR[WORKING,SRC]}/test_configuration"
    DIR[DTC,PERF,VPP]: "{DIR[DTC]}/vpp_performance_configuration"
    DIR[DTC,FUNC,VPP]: "{DIR[DTC]}/vpp_functional_configuration"

    # Detailed tests operational data
    DIR[DTO]: "{DIR[WORKING,SRC]}/test_operational_data"
    DIR[DTO,PERF,VPP]: "{DIR[DTO]}/vpp_performance_operational_data"

    # .css patch file to fix tables generated by Sphinx
    DIR[CSS_PATCH_FILE]: "{DIR[STATIC]}/theme_overrides.css"
    DIR[CSS_PATCH_FILE2]: "{DIR[WORKING,SRC,STATIC]}/theme_overrides.css"

  urls:
    URL[JENKINS,CSIT]: "https://jenkins.fd.io/view/csit/job"
    URL[JENKINS,HC]: "https://jenkins.fd.io/view/hc2vpp/job"

  make-dirs:
  # List the directories which are created while preparing the environment.
  # All directories MUST be defined in "paths" section.
  - "DIR[WORKING,DATA]"
  - "DIR[STATIC,VPP]"
  - "DIR[STATIC,DPDK]"
  - "DIR[STATIC,ARCH]"
  - "DIR[BUILD,LATEX]"
  - "DIR[WORKING,SRC]"
  - "DIR[WORKING,SRC,STATIC]"

  remove-dirs:
  # List the directories which are deleted while cleaning the environment.
  # All directories MUST be defined in "paths" section.
  #- "DIR[BUILD,HTML]"

  build-dirs:
  # List the directories where the results (build) is stored.
  # All directories MUST be defined in "paths" section.
  - "DIR[BUILD,HTML]"
  - "DIR[BUILD,LATEX]"
```

It is possible to use defined items in the definition of other items, e.g.:

```
DIR[WORKING,DATA]: "{DIR[WORKING]}/data"
```

will be automatically changed to

```
DIR[WORKING,DATA]: "_tmp/data"
```

Section: Configuration

This section specifies the groups of parameters which are repeatedly used in the elements defined later in the specification file. It has the following parts:

- data sets - Specification of data sets used later in element's specifications to define the input data.
- plot layouts - Specification of plot layouts used later in plots' specifications to define the plot layout.

The structure of the section "Configuration" is as follows (example):

```
-
  type: "configuration"
  data-sets:
    plot-vpp-throughput-latency:
      csit-vpp-perf-1710-all:
      - 11
      - 12
      - 13
      - 14
      - 15
      - 16
      - 17
      - 18
      - 19
      - 20
    vpp-perf-results:
      csit-vpp-perf-1710-all:
      - 20
      - 23
  plot-layouts:
    plot-throughput:
      xaxis:
        autorange: True
        autotick: False
        fixedrange: False
        gridcolor: "rgb(238, 238, 238)"
        linecolor: "rgb(238, 238, 238)"
        linewidth: 1
        showgrid: True
        showline: True
        showticklabels: True
        tickcolor: "rgb(238, 238, 238)"
        tickmode: "linear"
        title: "Indexed Test Cases"
        zeroline: False
      yaxis:
        gridcolor: "rgb(238, 238, 238)'"
        hoverformat: ".4s"
        linecolor: "rgb(238, 238, 238)"
        linewidth: 1
        range: []
        showgrid: True
        showline: True
        showticklabels: True
        tickcolor: "rgb(238, 238, 238)"
        title: "Packets Per Second [pps]"
        zeroline: False
      boxmode: "group"
```

(continues on next page)

(continued from previous page)

```
      boxgroupgap: 0.5
      autosize: False
      margin:
        t: 50
        b: 20
        l: 50
        r: 20
      showlegend: True
      legend:
        orientation: "h"
      width: 700
      height: 1000
```

The definitions from this sections are used in the elements, e.g.:

```
-
  type: "plot"
  title: "VPP Performance 64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
  algorithm: "plot_performance_box"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/64B-1t1c-l2-sel1-ndrdisc"
  data:
    "plot-vpp-throughput-latency"
  filter: "'64B' and ('BASE' or 'SCALE') and 'NDRDISC' and '1T1C' and ('L2BDMACSTAT' or 'L2BDMACLRN
↪' or 'L2XCFWD') and not 'VHOST'"
  parameters:
  - "throughput"
  - "parent"
  traces:
    hoverinfo: "x+y"
    boxpoints: "outliers"
    whiskerwidth: 0
  layout:
    title: "64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
    layout:
      "plot-throughput"
```

Section: Debug mode

This section is optional as it configures the debug mode. It is used if one does not want to download input data files and use local files instead.

If the debug mode is configured, the "input" section is ignored.

This section has the following parts:

- type: "debug" - says that this is the section "debug".
- general:
  - input-format - xml or zip.
  - extract - if "zip" is defined as the input format, this file is extracted from the zip file, otherwise this parameter is ignored.
- builds - list of builds from which the data is used. Must include a job name as a key and then a list of builds and their output files.

The structure of the section "Debug" is as follows (example):

```
-
  type: "debug"
```

(continues on next page)

(continued from previous page)

```
general:
  input-format: "zip"  # zip or xml
  extract: "robot-plugin/output.xml"  # Only for zip
builds:
  # The files must be in the directory DIR[WORKING,DATA]
  csit-dpdk-perf-1707-all:
  -
    build: 10
    file: "csit-dpdk-perf-1707-all__10.xml"
  -
    build: 9
    file: "csit-dpdk-perf-1707-all__9.xml"
  csit-vpp-functional-1707-ubuntu1604-virl:
  -
    build: lastSuccessfulBuild
    file: "csit-vpp-functional-1707-ubuntu1604-virl-lastSuccessfulBuild.xml"
  hc2vpp-csit-integration-1707-ubuntu1604:
  -
    build: lastSuccessfulBuild
    file: "hc2vpp-csit-integration-1707-ubuntu1604-lastSuccessfulBuild.xml"
  csit-vpp-perf-1707-all:
  -
    build: 16
    file: "csit-vpp-perf-1707-all__16__output.xml"
  -
    build: 17
    file: "csit-vpp-perf-1707-all__17__output.xml"
```

Section: Static

This section defines the static content which is stored in git and will be used as a source to generate the report.

This section has these parts:

- type: “static” - says that this section is the “static”.
- src-path - path to the static content.
- dst-path - destination path where the static content is copied and then processed.

```
-
  type: "static"
  src-path: "{DIR[RST]}"
  dst-path: "{DIR[WORKING,SRC]}"
```

Section: Input

This section defines the data used to generate elements. It is mandatory if the debug mode is not used.

This section has the following parts:

- type: “input” - says that this section is the “input”.
- general - parameters common to all builds:
  - file-name: file to be downloaded.
  - file-format: format of the downloaded file, “.zip” or “.xml” are supported.

  - download-path: path to be added to url pointing to the file, e.g.: "{job}/{build}/robot/report/*zip*/{filename}"; {job}, {build} and {filename} are replaced by proper values defined in this section.
  - extract: file to be extracted from downloaded zip file, e.g.: "output.xml"; if xml file is downloaded, this parameter is ignored.
- builds - list of jobs (keys) and numbers of builds which output data will be downloaded.

The structure of the section "Input" is as follows (example from 17.07 report):

```
-
  type: "input"  # Ignored in debug mode
  general:
    file-name: "robot-plugin.zip"
    file-format: ".zip"
    download-path: "{job}/{build}/robot/report/*zip*/{filename}"
    extract: "robot-plugin/output.xml"
  builds:
    csit-vpp-perf-1707-all:
    - 9
    - 10
    - 13
    - 14
    - 15
    - 16
    - 17
    - 18
    - 19
    - 21
    - 22
    csit-dpdk-perf-1707-all:
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10
    csit-vpp-functional-1707-ubuntu1604-virl:
    - lastSuccessfulBuild
    hc2vpp-csit-perf-master-ubuntu1604:
    - 8
    - 9
    hc2vpp-csit-integration-1707-ubuntu1604:
    - lastSuccessfulBuild
```

Section: Output

This section specifies which format(s) will be generated (html, pdf) and which versions will be generated for each format.

This section has the following parts:

- type: "output" - says that this section is the "output".
- format: html or pdf.
- version: defined for each format separately.

The structure of the section "Output" is as follows (example):

```
-
  type: "output"
  format:
    html:
    - full
    pdf:
    - full
    - minimal
```

TODO: define the names of versions

Content of "minimal" version

TODO: define the name and content of this version

Section: Table

This section defines a table to be generated. There can be 0 or more "table" sections.

This section has the following parts:

- type: "table" - says that this section defines a table.
- title: Title of the table.
- algorithm: Algorithm which is used to generate the table. The other parameters in this section must provide all information needed by the used algorithm.
- template: (optional) a .csv file used as a template while generating the table.
- output-file-ext: extension of the output file.
- output-file: file which the table will be written to.
- columns: specification of table columns:
  - title: The title used in the table header.
  - data: Specification of the data, it has two parts - command and arguments:
    * command:
      - template - take the data from template, arguments:
      - number of column in the template.
      - data - take the data from the input data, arguments:
      - jobs and builds which data will be used.
      - operation - performs an operation with the data already in the table, arguments:
      - operation to be done, e.g.: mean, stdev, relative_change (compute the relative change between two columns) and display number of data samples ~= number of test jobs. The operations are implemented in the utils.py TODO: Move from utils,py to e.g. operations.py
      - numbers of columns which data will be used (optional).
- data: Specify the jobs and builds which data is used to generate the table.
- filter: filter based on tags applied on the input data, if "template" is used, filtering is based on the template.
- parameters: Only these parameters will be put to the output data structure.

The structure of the section "Table" is as follows (example of "table_performance_improvements"):

```
-
  type: "table"
  title: "Performance improvements"
  algorithm: "table_performance_improvements"
  template: "{DIR[DTR,PERF,VPP,IMPRV]}/tmpl_performance_improvements.csv"
  output-file-ext: ".csv"
  output-file: "{DIR[DTR,PERF,VPP,IMPRV]}/performance_improvements"
  columns:
  -
    title: "VPP Functionality"
    data: "template 1"
  -
    title: "Test Name"
    data: "template 2"
  -
    title: "VPP-16.09 mean [Mpps]"
    data: "template 3"
  -
    title: "VPP-17.01 mean [Mpps]"
    data: "template 4"
  -
    title: "VPP-17.04 mean [Mpps]"
    data: "template 5"
  -
    title: "VPP-17.07 mean [Mpps]"
    data: "data csit-vpp-perf-1707-all mean"
  -
    title: "VPP-17.07 stdev [Mpps]"
    data: "data csit-vpp-perf-1707-all stdev"
  -
    title: "17.04 to 17.07 change [%]"
    data: "operation relative_change 5 4"
  data:
    csit-vpp-perf-1707-all:
    - 9
    - 10
    - 13
    - 14
    - 15
    - 16
    - 17
    - 18
    - 19
    - 21
  filter: "template"
  parameters:
  - "throughput"
```

Example of “table_details” which generates “Detailed Test Results - VPP Performance Results”:

```
-
  type: "table"
  title: "Detailed Test Results - VPP Performance Results"
  algorithm: "table_details"
  output-file-ext: ".csv"
  output-file: "{DIR[WORKING]}/vpp_performance_results"
  columns:
  -
    title: "Name"
    data: "data test_name"
  -
    title: "Documentation"
```

(continues on next page)

(continued from previous page)

```
    data: "data test_documentation"
  -
    title: "Status"
    data: "data test_msg"
  data:
    csit-vpp-perf-1707-all:
    - 17
  filter: "all"
  parameters:
  - "parent"
  - "doc"
  - "msg"
```

Example of “table_details” which generates “Test configuration - VPP Performance Test Configs”:

```
-
  type: "table"
  title: "Test configuration - VPP Performance Test Configs"
  algorithm: "table_details"
  output-file-ext: ".csv"
  output-file: "{DIR[WORKING]}/vpp_test_configuration"
  columns:
  -
    title: "Name"
    data: "data name"
  -
    title: "VPP API Test (VAT) Commands History - Commands Used Per Test Case"
    data: "data show-run"
  data:
    csit-vpp-perf-1707-all:
    - 17
  filter: "all"
  parameters:
  - "parent"
  - "name"
  - "show-run"
```

Section: Plot

This section defines a plot to be generated. There can be 0 or more “plot” sections.

This section has these parts:

- type: “plot” - says that this section defines a plot.
- title: Plot title used in the logs. Title which is displayed is in the section “layout”.
- output-file-type: format of the output file.
- output-file: file which the plot will be written to.
- algorithm: Algorithm used to generate the plot. The other parameters in this section must provide all information needed by plot.ly to generate the plot. For example:
  - traces
  - layout
  - These parameters are transparently passed to plot.ly.
- data: Specify the jobs and numbers of builds which data is used to generate the plot.
- filter: filter applied on the input data.

- parameters: Only these parameters will be put to the output data structure.

The structure of the section "Plot" is as follows (example of a plot showing throughput in a chart box-with-whiskers):

```
-
  type: "plot"
  title: "VPP Performance 64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
  algorithm: "plot_performance_box"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/64B-1t1c-l2-sel1-ndrdisc"
  data:
    csit-vpp-perf-1707-all:
    - 9
    - 10
    - 13
    - 14
    - 15
    - 16
    - 17
    - 18
    - 19
    - 21
  # Keep this formatting, the filter is enclosed with " (quotation mark) and
  # each tag is enclosed with ' (apostrophe).
  filter: "'64B' and 'BASE' and 'NDRDISC' and '1T1C' and ('L2BDMACSTAT' or 'L2BDMACLRN' or 'L2XCFWD
↪') and not 'VHOST'"
  parameters:
  - "throughput"
  - "parent"
  traces:
    hoverinfo: "x+y"
    boxpoints: "outliers"
    whiskerwidth: 0
  layout:
    title: "64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
    xaxis:
      autorange: True
      autotick: False
      fixedrange: False
      gridcolor: "rgb(238, 238, 238)"
      linecolor: "rgb(238, 238, 238)"
      linewidth: 1
      showgrid: True
      showline: True
      showticklabels: True
      tickcolor: "rgb(238, 238, 238)"
      tickmode: "linear"
      title: "Indexed Test Cases"
      zeroline: False
    yaxis:
      gridcolor: "rgb(238, 238, 238)'"
      hoverformat: ".4s"
      linecolor: "rgb(238, 238, 238)"
      linewidth: 1
      range: []
      showgrid: True
      showline: True
      showticklabels: True
      tickcolor: "rgb(238, 238, 238)"
      title: "Packets Per Second [pps]"
      zeroline: False
    boxmode: "group"
```

(continues on next page)

(continued from previous page)

```
    boxgroupgap: 0.5
    autosize: False
    margin:
      t: 50
      b: 20
      l: 50
      r: 20
    showlegend: True
    legend:
      orientation: "h"
    width: 700
    height: 1000
```

The structure of the section "Plot" is as follows (example of a plot showing latency in a box chart):

```
-
  type: "plot"
  title: "VPP Latency 64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
  algorithm: "plot_latency_box"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/64B-1t1c-l2-sel1-ndrdisc-lat50"
  data:
    csit-vpp-perf-1707-all:
    - 9
    - 10
    - 13
    - 14
    - 15
    - 16
    - 17
    - 18
    - 19
    - 21
  filter: "'64B' and 'BASE' and 'NDRDISC' and '1T1C' and ('L2BDMACSTAT' or 'L2BDMACLRN' or 'L2XCFWD
↪') and not 'VHOST'"
  parameters:
  - "latency"
  - "parent"
  traces:
    boxmean: False
  layout:
    title: "64B-1t1c-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
    xaxis:
      autorange: True
      autotick: False
      fixedrange: False
      gridcolor: "rgb(238, 238, 238)"
      linecolor: "rgb(238, 238, 238)"
      linewidth: 1
      showgrid: True
      showline: True
      showticklabels: True
      tickcolor: "rgb(238, 238, 238)"
      tickmode: "linear"
      title: "Indexed Test Cases"
      zeroline: False
    yaxis:
      gridcolor: "rgb(238, 238, 238)'"
      hoverformat: ""
      linecolor: "rgb(238, 238, 238)"
      linewidth: 1
```

(continues on next page)

(continued from previous page)

```
      range: []
      showgrid: True
      showline: True
      showticklabels: True
      tickcolor: "rgb(238, 238, 238)"
      title: "Latency min/avg/max [uSec]"
      zeroline: False
    boxmode: "group"
    boxgroupgap: 0.5
    autosize: False
    margin:
      t: 50
      b: 20
      l: 50
      r: 20
    showlegend: True
    legend:
      orientation: "h"
    width: 700
    height: 1000
```

The structure of the section "Plot" is as follows (example of a plot showing VPP HTTP server performance in a box chart with pre-defined data "plot-vpp-httlp-server-performance" set and plot layout "plot-cps"):

```
-
  type: "plot"
  title: "VPP HTTP Server Performance"
  algorithm: "plot_http_server_perf_box"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/http-server-performance-cps"
  data:
    "plot-vpp-httlp-server-performance"
  # Keep this formatting, the filter is enclosed with " (quotation mark) and
  # each tag is enclosed with ' (apostrophe).
  filter: "'HTTP' and 'TCP_CPS'"
  parameters:
  - "result"
  - "name"
  traces:
    hoverinfo: "x+y"
    boxpoints: "outliers"
    whiskerwidth: 0
  layout:
    title: "VPP HTTP Server Performance"
    layout:
      "plot-cps"
```

Section: file

This section defines a file to be generated. There can be 0 or more "file" sections.

This section has the following parts:

- type: "file" - says that this section defines a file.
- title: Title of the table.
- algorithm: Algorithm which is used to generate the file. The other parameters in this section must provide all information needed by the used algorithm.
- output-file-ext: extension of the output file.

- output-file: file which the file will be written to.
- file-header: The header of the generated .rst file.
- dir-tables: The directory with the tables.
- data: Specify the jobs and builds which data is used to generate the table.
- filter: filter based on tags applied on the input data, if "all" is used, no filtering is done.
- parameters: Only these parameters will be put to the output data structure.
- chapters: the hierarchy of chapters in the generated file.
- start-level: the level of the the top-level chapter.

The structure of the section "file" is as follows (example):

```
-
  type: "file"
  title: "VPP Performance Results"
  algorithm: "file_test_results"
  output-file-ext: ".rst"
  output-file: "{DIR[DTR,PERF,VPP]}/vpp_performance_results"
  file-header: "\n.. |br| raw:: html\n\n    <br />\n\n\n.. |prein| raw:: html\n\n    <pre>\n\n\n..␣
↪|preout| raw:: html\n\n    </pre>\n\n"
  dir-tables: "{DIR[DTR,PERF,VPP]}"
  data:
    csit-vpp-perf-1707-all:
    - 22
  filter: "all"
  parameters:
  - "name"
  - "doc"
  - "level"
  data-start-level: 2  # 0, 1, 2, ...
  chapters-start-level: 2  # 0, 1, 2, ...
```

Static content

- Manually created / edited files.
- .rst files, static .csv files, static pictures (.svg), ...
- Stored in CSIT git repository.

No more details about the static content in this document.

Data to process

The PAL processes tests results and other information produced by Jenkins jobs. The data are now stored as robot results in Jenkins (TODO: store the data in nexus) either as .zip and / or .xml files.

4.3.3 Data processing

As the first step, the data are downloaded and stored locally (typically on a Jenkins slave). If .zip files are used, the given .xml files are extracted for further processing.

Parsing of the .xml files is performed by a class derived from "robot.api.ResultVisitor", only necessary methods are overridden. All and only necessary data is extracted from .xml file and stored in a structured form.

The parsed data are stored as the multi-indexed pandas.Series data type. Its structure is as follows:

```
<job name>
  <build>
    <metadata>
    <suites>
    <tests>
```

"job name", "build", "metadata", "suites", "tests" are indexes to access the data. For example:

```
data =

job 1 name:
  build 1:
    metadata: metadata
    suites: suites
    tests: tests
  ...
  build N:
    metadata: metadata
    suites: suites
    tests: tests
...
job M name:
  build 1:
    metadata: metadata
    suites: suites
    tests: tests
  ...
  build N:
    metadata: metadata
    suites: suites
    tests: tests
```

Using indexes data["job 1 name"]["build 1"]["tests"] (e.g.: data["csit-vpp-perf-1704-all"]["17"]["tests"]) we get a list of all tests with all tests data.

Data will not be accessible directly using indexes, but using getters and filters.

Structure of metadata:

```
"metadata": {
    "version": "VPP version",
    "job": "Jenkins job name"
    "build": "Information about the build"
},
```

Structure of suites:

```
"suites": {
    "Suite name 1": {
        "doc": "Suite 1 documentation"
        "parent": "Suite 1 parent"
    }
    "Suite name N": {
        "doc": "Suite N documentation"
        "parent": "Suite N parent"
    }
```

Structure of tests:

Performance tests:

```
"tests": {
    "ID": {
```

(continues on next page)

(continued from previous page)

```
        "name": "Test name",
        "parent": "Name of the parent of the test",
        "doc": "Test documentation"
        "msg": "Test message"
        "tags": ["tag 1", "tag 2", "tag n"],
        "type": "PDR" | "NDR",
        "throughput": {
            "value": int,
            "unit": "pps" | "bps" | "percentage"
        },
        "latency": {
            "direction1": {
                "100": {
                    "min": int,
                    "avg": int,
                    "max": int
                },
                "50": {  # Only for NDR
                    "min": int,
                    "avg": int,
                    "max": int
                },
                "10": {  # Only for NDR
                    "min": int,
                    "avg": int,
                    "max": int
                }
            },
            "direction2": {
                "100": {
                    "min": int,
                    "avg": int,
                    "max": int
                },
                "50": {  # Only for NDR
                    "min": int,
                    "avg": int,
                    "max": int
                },
                "10": {  # Only for NDR
                    "min": int,
                    "avg": int,
                    "max": int
                }
            }
        },
        "lossTolerance": "lossTolerance"  # Only for PDR
        "vat-history": "DUT1 and DUT2 VAT History"
        },
        "show-run": "Show Run"
    },
    "ID" {
        # next test
    }
```

Functional tests:

```
"tests": {
    "ID": {
        "name": "Test name",
        "parent": "Name of the parent of the test",
```

(continues on next page)

(continued from previous page)

```
        "doc": "Test documentation"
        "msg": "Test message"
        "tags": ["tag 1", "tag 2", "tag n"],
        "vat-history": "DUT1 and DUT2 VAT History"
        "show-run": "Show Run"
        "status": "PASS" | "FAIL"
    },
    "ID" {
        # next test
    }
}
```

Note: ID is the lowercase full path to the test.

Data filtering

The first step when generating an element is getting the data needed to construct the element. The data are filtered from the processed input data.

The data filtering is based on:

- job name(s).
- build number(s).
- tag(s).
- required data - only this data is included in the output.

WARNING: The filtering is based on tags, so be careful with tagging.

For example, the element which specification includes:

```
data:
  csit-vpp-perf-1707-all:
  - 9
  - 10
  - 13
  - 14
  - 15
  - 16
  - 17
  - 18
  - 19
  - 21
filter:
  - "'64B' and 'BASE' and 'NDRDISC' and '1T1C' and ('L2BDMACSTAT' or 'L2BDMACLRN' or 'L2XCFWD') and␣
↪not 'VHOST'"
```

will be constructed using data from the job “csit-vpp-perf-1707-all”, for all listed builds and the tests with the list of tags matching the filter conditions.

The output data structure for filtered test data is:

```
- job 1
  - build 1
    - test 1
      - parameter 1
      - parameter 2
      ...
      - parameter n
    ...
    - test n
```

(continues on next page)

(continued from previous page)

```
    ...
  ...
  - build n
...
- job n
```

Data analytics

Data analytics part implements:

- methods to compute statistical data from the filtered input data.
- trending.

Throughput Speedup Analysis - Multi-Core with Multi-Threading

Throughput Speedup Analysis (TSA) calculates throughput speedup ratios for tested 1-, 2- and 4-core multi-threaded VPP configurations using the following formula:

```
                            N_core_throughput
N_core_throughput_speedup = -----------------
                            1_core_throughput
```

Multi-core throughput speedup ratios are plotted in grouped bar graphs for throughput tests with 64B/78B frame size, with number of cores on X-axis and speedup ratio on Y-axis.

For better comparison multiple test results' data sets are plotted per each graph:

- graph type: grouped bars;
- graph X-axis: (testcase index, number of cores);
- graph Y-axis: speedup factor.

Subset of existing performance tests is covered by TSA graphs.

Model for TSA:

```
-
  type: "plot"
  title: "TSA: 64B-*-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
  algorithm: "plot_throughput_speedup_analysis"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/10ge2p1x520-64B-l2-tsa-ndrdisc"
  data:
    "plot-throughput-speedup-analysis"
  filter: "'NIC_Intel-X520-DA2' and '64B' and 'BASE' and 'NDRDISC' and ('L2BDMACSTAT' or 'L2BDMACLRN
↪' or 'L2XCFWD') and not 'VHOST'"
  parameters:
  - "throughput"
  - "parent"
  - "tags"
  layout:
    title: "64B-*-(eth|dot1q|dot1ad)-(l2xcbase|l2bdbasemaclrn)-ndrdisc"
    layout:
      "plot-throughput-speedup-analysis"
```

Comparison of results from two sets of the same test executions

This algorithm enables comparison of results coming from two sets of the same test executions. It is used to quantify performance changes across all tests after test environment changes e.g. Operating System upgrades/patches, Hardware changes.

It is assumed that each set of test executions includes multiple runs of the same tests, 10 or more, to verify test results repeatibility and to yield statistically meaningful results data.

Comparison results are presented in a table with a specified number of the best and the worst relative changes between the two sets. Following table columns are defined:

- name of the test;
- throughput mean values of the reference set;
- throughput standard deviation of the reference set;
- throughput mean values of the set to compare;
- throughput standard deviation of the set to compare;
- relative change of the mean values.

The model

The model specifies:

- type: “table” - means this section defines a table.
- title: Title of the table.
- algorithm: Algorithm which is used to generate the table. The other parameters in this section must provide all information needed by the used algorithm.
- output-file-ext: Extension of the output file.
- output-file: File which the table will be written to.
- reference - the builds which are used as the reference for comparison.
- compare - the builds which are compared to the reference.
- data: Specify the sources, jobs and builds, providing data for generating the table.
- filter: Filter based on tags applied on the input data, if “template” is used, filtering is based on the template.
- parameters: Only these parameters will be put to the output data structure.
- nr-of-tests-shown: Number of the best and the worst tests presented in the table. Use 0 (zero) to present all tests.

Example:

```
-
  type: "table"
  title: "Performance comparison"
  algorithm: "table_perf_comparison"
  output-file-ext: ".csv"
  output-file: "{DIR[DTR,PERF,VPP,IMPRV]}/vpp_performance_comparison"
  reference:
    title: "csit-vpp-perf-1801-all - 1"
    data:
      csit-vpp-perf-1801-all:
      - 1
      - 2
  compare:
    title: "csit-vpp-perf-1801-all - 2"
```

(continues on next page)

(continued from previous page)

```
    data:
      csit-vpp-perf-1801-all:
      - 1
      - 2
  data:
    "vpp-perf-comparison"
  filter: "all"
  parameters:
  - "name"
  - "parent"
  - "throughput"
  nr-of-tests-shown: 20
```

Advanced data analytics

In the future advanced data analytics (ADA) will be added to analyze the telemetry data collected from SUT telemetry sources and correlate it to performance test results.

TODO

- describe the concept of ADA.
- add specification.

4.3.4 Data presentation

Generates the plots and tables according to the report models per specification file. The elements are generated using algorithms and data specified in their models.

Tables

- tables are generated by algorithms implemented in PAL, the model includes the algorithm and all necessary information.
- output format: csv
- generated tables are stored in specified directories and linked to .rst files.

Plots

- plot.ly[109] is currently used to generate plots, the model includes the type of plot and all the necessary information to render it.
- output format: html.
- generated plots are stored in specified directories and linked to .rst files.

4.3.5 Report generation

Report is generated using Sphinx and Read_the_Docs template. PAL generates html and pdf formats. It is possible to define the content of the report by specifying the version (TODO: define the names and content of versions).

[109] https://plot.ly/

The process

1. Read the specification.
2. Read the input data.
3. Process the input data.
4. For element (plot, table, file) defined in specification:
   a. Get the data needed to construct the element using a filter.
   b. Generate the element.
   c. Store the element.
5. Generate the report.
6. Store the report (Nexus).

The process is model driven. The elements' models (tables, plots, files and report itself) are defined in the specification file. Script reads the elements' models from specification file and generates the elements.

It is easy to add elements to be generated in the report. If a new type of an element is required, only a new algorithm needs to be implemented and integrated.

4.3.6 Continuous Performance Measurements and Trending

Performance analysis and trending execution sequence:

CSIT PA runs performance analysis, change detection and trending using specified trend analysis metrics over the rolling window of last <N> sets of historical measurement data. PA is defined as follows:

1. PA job triggers:
   1. By PT job at its completion.
   2. Manually from Jenkins UI.
2. Download and parse archived historical data and the new data:
   1. New data from latest PT job is evaluated against the rolling window of <N> sets of historical data.
   2. Download RF output.xml files and compressed archived data.
   3. Parse out the data filtering test cases listed in PA specification (part of CSIT PAL specification file).
3. Calculate trend metrics for the rolling window of <N> sets of historical data:
   1. Calculate quartiles Q1, Q2, Q3.
   2. Trim outliers using IQR.
   3. Calculate TMA and TMSD.
   4. Calculate normal trending range per test case based on TMA and TMSD.
4. Evaluate new test data against trend metrics:
   1. If within the range of (TMA +/- 3*TMSD) => Result = Pass, Reason = Normal.
   2. If below the range => Result = Fail, Reason = Regression.
   3. If above the range => Result = Pass, Reason = Progression.
5. Generate and publish results
   1. Relay evaluation result to job result.

2. Generate a new set of trend analysis summary graphs and drill-down graphs.
   1. Summary graphs to include measured values with Normal, Progression and Regression markers. MM shown in the background if possible.
   2. Drill-down graphs to include MM, TMA and TMSD.
3. Publish trend analysis graphs in html format on https://docs.fd.io/csit/master/trending/.

Parameters to specify:

General section - parameters common to all plots:

- type: "cpta";
- title: The title of this section;
- output-file-type: only ".html" is supported;
- output-file: path where the generated files will be stored.

Plots section:

- plot title;
- output file name;
- input data for plots;
  - job to be monitored - the Jenkins job which results are used as input data for this test;
  - builds used for trending plot(s) - specified by a list of build numbers or by a range of builds defined by the first and the last build number;
- tests to be displayed in the plot defined by a filter;
- list of parameters to extract from the data;
- plot layout

Example:

```
-
  type: "cpta"
  title: "Continuous Performance Trending and Analysis"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/cpta"
  plots:

    - title: "VPP 1T1C L2 64B Packet Throughput - Trending"
      output-file-name: "l2-1t1c-x520"
      data: "plot-performance-trending-vpp"
      filter: "'NIC_Intel-X520-DA2' and 'MRR' and '64B' and ('BASE' or 'SCALE') and '1T1C' and (
↪'L2BDMACSTAT' or 'L2BDMACLRN' or 'L2XCFWD') and not 'VHOST' and not 'MEMIF'"
      parameters:
      - "result"
      layout: "plot-cpta-vpp"

    - title: "DPDK 4T4C IMIX MRR Trending"
      output-file-name: "dpdk-imix-4t4c-xl710"
      data: "plot-performance-trending-dpdk"
      filter: "'NIC_Intel-XL710' and 'IMIX' and 'MRR' and '4T4C' and 'DPDK'"
      parameters:
      - "result"
      layout: "plot-cpta-dpdk"
```

The Dashboard

Performance dashboard tables provide the latest VPP throughput trend, trend compliance and detected anomalies, all on a per VPP test case basis. The Dashboard is generated as three tables for 1t1c, 2t2c and 4t4c MRR tests.

At first, the .csv tables are generated (only the table for 1t1c is shown):

```
-
  type: "table"
  title: "Performance trending dashboard"
  algorithm: "table_perf_trending_dash"
  output-file-ext: ".csv"
  output-file: "{DIR[STATIC,VPP]}/performance-trending-dashboard-1t1c"
  data: "plot-performance-trending-all"
  filter: "'MRR' and '1T1C'"
  parameters:
  - "name"
  - "parent"
  - "result"
  ignore-list:
  - "tests.vpp.perf.l2.10ge2p1x520-eth-l2bdscale1mmaclrn-mrr.tc01-64b-1t1c-eth-l2bdscale1mmaclrn-
↪ndrdisc"
  outlier-const: 1.5
  window: 14
  evaluated-window: 14
  long-trend-window: 180
```

Then, html tables stored inside .rst files are generated:

```
-
  type: "table"
  title: "HTML performance trending dashboard 1t1c"
  algorithm: "table_perf_trending_dash_html"
  input-file: "{DIR[STATIC,VPP]}/performance-trending-dashboard-1t1c.csv"
  output-file: "{DIR[STATIC,VPP]}/performance-trending-dashboard-1t1c.rst"
```

4.3.7 Root Cause Analysis

Root Cause Analysis (RCA) by analysing archived performance results – re-analyse available data for specified:

- range of jobs builds,
- set of specific tests and
- PASS/FAIL criteria to detect performance change.

In addition, PAL generates trending plots to show performance over the specified time interval.

Root Cause Analysis - Option 1: Analysing Archived VPP Results

It can be used to speed-up the process, or when the existing data is sufficient. In this case, PAL uses existing data saved in Nexus, searches for performance degradations and generates plots to show performance over the specified time interval for the selected tests.

Execution Sequence

1. Download and parse archived historical data and the new data.
2. Calculate trend metrics.

3. Find regression / progression.
4. Generate and publish results:
   1. Summary graphs to include measured values with Progression and Regression markers.
   2. List the DUT build(s) where the anomalies were detected.

CSIT PAL Specification

- What to test:
  - first build (Good); specified by the Jenkins job name and the build number
  - last build (Bad); specified by the Jenkins job name and the build number
  - step (1..n).
- Data:
  - tests of interest; list of tests (full name is used) which results are used

Example:

```
TODO
```

4.3.8 API

List of modules, classes, methods and functions

```
specification_parser.py

    class Specification

        Methods:
            read_specification
            set_input_state
            set_input_file_name

        Getters:
            specification
            environment
            debug
            is_debug
            input
            builds
            output
            tables
            plots
            files
            static


input_data_parser.py

    class InputData

        Methods:
            read_data
            filter_data

        Getters:
```

(continues on next page)

(continued from previous page)

```
            data
            metadata
            suites
            tests


environment.py

    Functions:
        clean_environment

    class Environment

        Methods:
            set_environment

        Getters:
            environment


input_data_files.py

    Functions:
        download_data_files
        unzip_files


generator_tables.py

    Functions:
        generate_tables

    Functions implementing algorithms to generate particular types of
    tables (called by the function "generate_tables"):
        table_details
        table_performance_improvements


generator_plots.py

    Functions:
        generate_plots

    Functions implementing algorithms to generate particular types of
    plots (called by the function "generate_plots"):
        plot_performance_box
        plot_latency_box


generator_files.py

    Functions:
        generate_files

    Functions implementing algorithms to generate particular types of
    files (called by the function "generate_files"):
        file_test_results


report.py
```

(continues on next page)

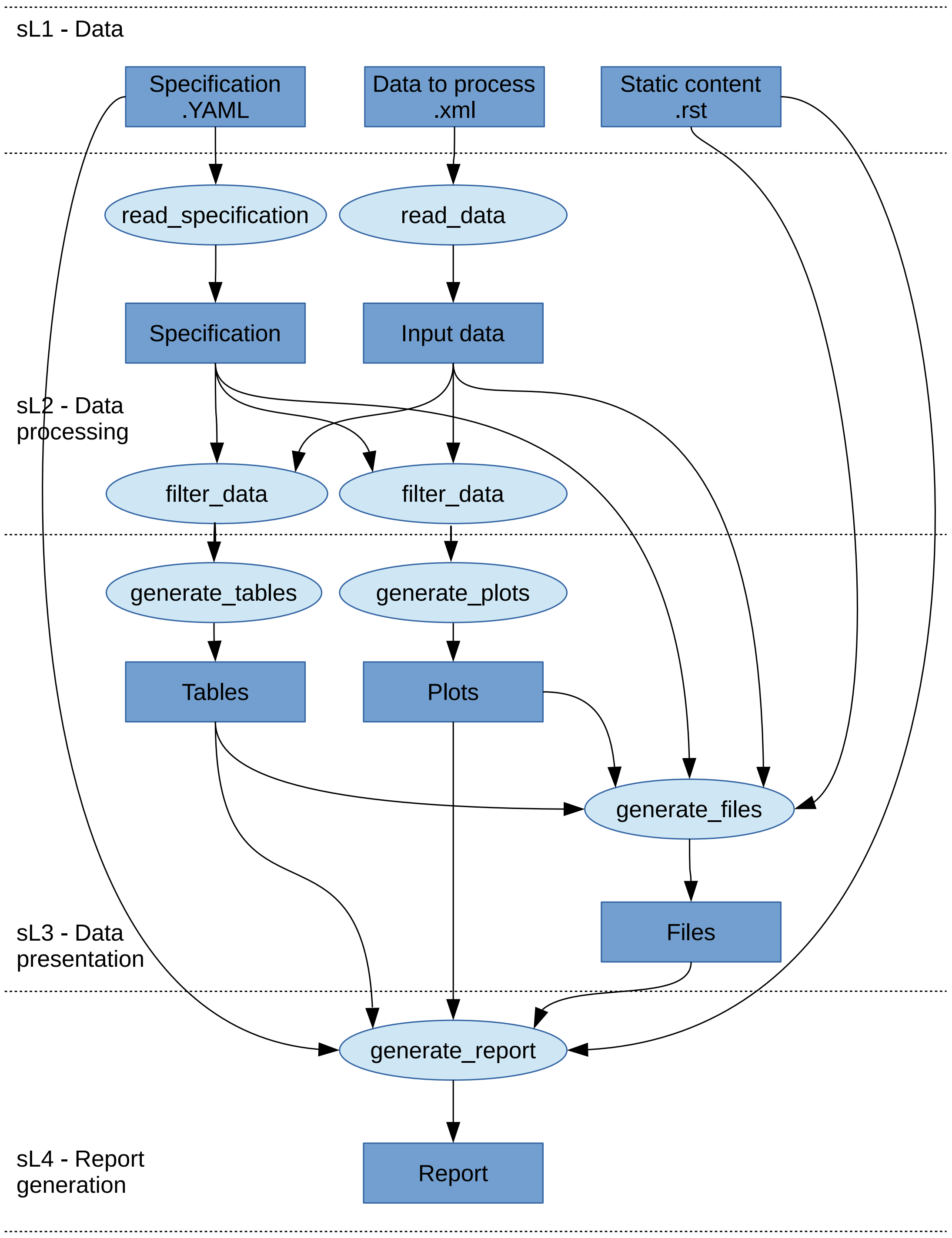
(continued from previous page)

```
    Functions:
        generate_report

    Functions implementing algorithms to generate particular types of
    report (called by the function "generate_report"):
        generate_html_report
        generate_pdf_report

    Other functions called by the function "generate_report":
        archive_input_data
        archive_report
```

PAL functional diagram



How to add an element

Element can be added by adding it's model to the specification file. If the element is to be generated by an existing algorithm, only it's parameters must be set.

If a brand new type of element needs to be added, also the algorithm must be implemented. Element generation algorithms are implemented in the files with names starting with "generator" prefix. The name of the function implementing the algorithm and the name of algorithm in the specification file have to be the same.

4.4 CSIT RF Tags Descriptions

All CSIT test cases are labelled with Robot Framework tags used to allow for easy test case type identification, test case grouping and selection for execution. Following sections list currently used CSIT TAGs and their documentation based on the content of tag documentation rst file[110].

4.4.1 Testbed Topology Tags

2_NODE_DOUBLE_LINK_TOPO

2 nodes connected in a circular topology with two links interconnecting the devices.

2_NODE_SINGLE_LINK_TOPO

2 nodes connected in a circular topology with at least one link interconnecting devices.

3_NODE_DOUBLE_LINK_TOPO

3 nodes connected in a circular topology with two links interconnecting the devices.

3_NODE_SINGLE_LINK_TOPO

3 nodes connected in a circular topology with at least one link interconnecting devices.

4.4.2 Objective Tags

SKIP_PATCH

Test case(s) marked to not run in case of vpp-csit-verify (i.e. VPP patch) and csit-vpp-verify jobs (i.e. CSIT patch).

SKIP_VPP_PATCH

Test case(s) marked to not run in case of vpp-csit-verify (i.e. VPP patch).

4.4.3 Environment Tags

HW_ENV

DUTs and TGs are running on bare metal.

VM_ENV

DUTs and TGs are running in virtual environment.

VPP_VM_ENV

DUTs with VPP and capable of running Virtual Machine.

[110] https://git.fd.io/csit/tree/docs/tag_documentation.rst?h=rls1908_2

4.4.4 NIC Model Tags

NIC_Intel-X520-DA2

Intel X520-DA2 NIC.

NIC_Intel-XL710

Intel XL710 NIC.

NIC_Intel-X710

Intel X710 NIC.

NIC_Intel-XXV710

Intel XXV710 NIC.

NIC_Cisco-VIC-1227

VIC-1227 by Cisco.

NIC_Cisco-VIC-1385

VIC-1385 by Cisco.

4.4.5 Scaling Tags

FIB_20K

2x10,000 entries in single fib table

FIB_200K

2x100,000 entries in single fib table

FIB_2M

2x1,000,000 entries in single fib table

L2BD_1

Test with 1 L2 bridge domain.

L2BD_10

Test with 10 L2 bridge domains.

L2BD_100

Test with 100 L2 bridge domains.

L2BD_1K

Test with 1000 L2 bridge domains.

VLAN_1

Test with 1 VLAN sub-interface.

VLAN_10

Test with 10 VLAN sub-interfaces.

VLAN_100

Test with 100 VLAN sub-interfaces.

VLAN_1K

Test with 1000 VLAN sub-interfaces.

VXLAN_1

Test with 1 VXLAN tunnel.

VXLAN_10

Test with 10 VXLAN tunnels.

VXLAN_100

Test with 100 VXLAN tunnels.

VXLAN_1K

Test with 1000 VXLAN tunnels.

TNL_{t}

IPSec in tunnel mode - {t} tunnels.

SRC_USER_1

Traffic flow with 1 unique IP (users) in one direction.

SRC_USER_10

Traffic flow with 10 unique IPs (users) in one direction.

SRC_USER_100

Traffic flow with 100 unique IPs (users) in one direction.

SRC_USER_1000

Traffic flow with 1000 unique IPs (users) in one direction.

SRC_USER_2000

Traffic flow with 2000 unique IPs (users) in one direction.

SRC_USER_4000

Traffic flow with 4000 unique IPs (users) in one direction.

100_FLOWS

Traffic stream with 100 unique flows (10 IPs/users x 10 UDP ports) in one direction.

10k_FLOWS

Traffic stream with 10 000 unique flows (10 IPs/users x 1000 UDP ports) in one direction.

100k_FLOWS

Traffic stream with 100 000 unique flows (100 IPs/users x 1000 UDP ports) in one direction.

4.4.6 Test Category Tags

FUNCTEST

All functional test cases.

PERFTEST

All performance test cases.

4.4.7 Performance Type Tags

NDRPDR

Single test finding both No Drop Rate and Partial Drop Rate simultaneously. The search is done by optimized algorithm which performs multiple trial runs at different durations and transmit rates. The results come from the final trials, which have duration of 30 seconds.

MRR

Performance tests where TG sends the traffic at maximum rate (line rate) and reports total sent/received packets over trial duration. The result is an average of 10 trials of 1 second duration.

SOAK

Performance tests using PLRsearch to find the critical load.

RECONF

Performance tests aimed to measure lost packets (time) when performing reconfiguration while full throughput offered load is applied.

4.4.8 Ethernet Frame Size Tags

These are describing the traffic offered by Traffic Generator, "primary" traffic in case of asymmetric load. For traffic between DUTs, or for "secondary" traffic, see ${overhead} value.

64B

64B frames used for test. Generic ethernet or IPv4.

78B

78B frames used for test. Ipv6.

114B

114B frames used for test. IPv4+vxlan.

118B

118B frames used for test. Dot1q+IPv4+vxlan.

IMIX

IMIX frame sequence (28x 64B, 16x 570B, 4x 1518B) used for test.

1460B

1460B frames used for test.

1480B

1480B frames used for test.

1514B

1514B frames used for test.

1518B

1518B frames used for test.

9000B

9000B frames used for test.

4.4.9 Test Type Tags

BASE

Baseline test cases, no encapsulation, no feature(s) configured in tests.

IP4BASE

IPv4 baseline test cases, no encapsulation, no feature(s) configured in tests.

IP6BASE

IPv6 baseline test cases, no encapsulation, no feature(s) configured in tests.

L2XCBASE

L2XC baseline test cases, no encapsulation, no feature(s) configured in tests.

L2BDBASE

L2BD baseline test cases, no encapsulation, no feature(s) configured in tests.

L2PATCH

L2PATCH baseline test cases, no encapsulation, no feature(s) configured in tests.

SCALE

Scale test cases.

ENCAP

Test cases where encapsulation is used. Use also encapsulation tag(s).

FEATURE

At least one feature is configured in test cases. Use also feature tag(s).

TLDK

Functional test cases for TLDK.

DMM

Functional test cases for DMM.

TCP

Tests which use TCP.

TCP_CPS

Performance tests which measure connections per second using http requests.

TCP_RPS

Performance tests which measure requests per second using http requests.

HTTP

Tests which use HTTP.

NF_DENSITY

Performance tests that measure throughput of multiple VNF and CNF service topologies at different service densities.

4.4.10 NF Service Density Tags

CHAIN

NF service density tests with VNF or CNF service chain topology(ies).

PIPE

NF service density tests with CNF service pipeline topology(ies).

NF_L3FWDIP4

NF service density tests with DPDK l3fwd IPv4 routing as NF workload.

NF_VPPIP4

NF service density tests with VPP IPv4 routing as NF workload.

{r}R{c}C

Service density matrix locator {r}R{c}C, {r}Row denoting number of service instances, {c}Column denoting number of NFs per service instance. {r}=(1,2,4,6,8,10), {c}=(1,2,4,6,8,10).

{n}VM{t}T

Service density {n}VM{t}T, {n}Number of NF Qemu VMs, {t}Number of threads per NF.

{n}DCRt}T

Service density {n}DCR{t}T, {n}Number of NF Docker containers, {t}Number of threads per NF.

{n}_ADDED_CHAINS

{n}Number of chains (or pipelines) added (and/or removed) during RECONF test.

4.4.11 Forwarding Mode Tags

L2BDMACSTAT

VPP L2 bridge-domain, L2 MAC static.

L2BDMACLRN

VPP L2 bridge-domain, L2 MAC learning.

L2XCFWD

VPP L2 point-to-point cross-connect.

IP4FWD

VPP IPv4 routed forwarding.

IP6FWD

VPP IPv6 routed forwarding.

4.4.12 Underlay Tags

IP4UNRLAY

IPv4 underlay.

IP6UNRLAY

IPv6 underlay.

MPLSUNRLAY

MPLS underlay.

4.4.13 Overlay Tags

L2OVRLAY

L2 overlay.

IP4OVRLAY

IPv4 overlay (IPv4 payload).

IP6OVRLAY

IPv6 overlay (IPv6 payload).

4.4.14 Tagging Tags

DOT1Q

All test cases with dot1q.

DOT1AD

All test cases with dot1ad.

4.4.15 Encapsulation Tags

ETH

All test cases with base Ethernet (no encapsulation).

LISP

All test cases with LISP.

LISPGPE

All test cases with LISP-GPE.

VXLAN

All test cases with Vxlan.

VXLANGPE

All test cases with VXLAN-GPE.

GRE

All test cases with GRE.

IPSEC

All test cases with IPSEC.

SRv6

All test cases with Segment routing over IPv6 dataplane.

SRv6_1SID

All SRv6 test cases with single SID.

SRv6_2SID_DECAP

All SRv6 test cases with two SIDs and with decapsulation.

SRv6_2SID_NODECAP

All SRv6 test cases with two SIDs and without decapsulation.

4.4.16 Interface Tags

PHY

All test cases which use physical interface(s).

VHOST

All test cases which uses VHOST.

VHOST_256

All test cases which uses VHOST with qemu queue size set to 256.

VHOST_1024

All test cases which uses VHOST with qemu queue size set to 1024.

CFS_OPT

All test cases which uses VM with optimised scheduler policy.

TUNTAP

All test cases which uses TUN and TAP.

AFPKT

All test cases which uses AFPKT.

NETMAP

All test cases which uses Netmap.

MEMIF

All test cases which uses Memif.

SINGLE_MEMIF

All test cases which uses only single Memif connection per DUT. One DUT instance is running in container having one physical interface exposed to container.

LBOND

All test cases which uses link bonding (BondEthernet interface).

LBOND_DPDK

All test cases which uses DPDK link bonding.

LBOND_VPP

All test cases which uses VPP link bonding.

LBOND_MODE_XOR

All test cases which uses link bonding with mode XOR.

LBOND_MODE_LACP

All test cases which uses link bonding with mode LACP.

LBOND_LB_L34

All test cases which uses link bonding with load-balance mode l34.

LBOND_1L

All test cases which uses one link for link bonding.

LBOND_2L

All test cases which uses two links for link bonding.

DRV_AVF

All test cases which uses Intel Adaptive Virtual Function (AVF) device plugin for VPP. This plugins provides native device support for Intel AVF. AVF is driver specification for current and future Intel Virtual Function devices. In essence, today this driver can be used only with Intel XL710 / X710 / XXV710 adapters.

4.4.17 Feature Tags

IACLDST

iACL destination.

COPWHLIST

COP whitelist.

NAT44

NAT44 configured and tested.

NAT64

NAT44 configured and tested.

ACL

ACL plugin configured and tested.

IACL

ACL plugin configured and tested on input path.

OACL

ACL plugin configured and tested on output path.

ACL_STATELESS

ACL plugin configured and tested in stateless mode (permit action).

ACL_STATEFUL

ACL plugin configured and tested in stateful mode (permit+reflect action).

ACL1

ACL plugin configured and tested with 1 not-hitting ACE.

ACL10

ACL plugin configured and tested with 10 not-hitting ACEs.

ACL50

ACL plugin configured and tested with 50 not-hitting ACEs.

SRv6_PROXY

SRv6 endpoint to SR-unaware appliance via proxy.

SRv6_PROXY_STAT

SRv6 endpoint to SR-unaware appliance via static proxy.

SRv6_PROXY_DYN

SRv6 endpoint to SR-unaware appliance via dynamic proxy.

SRv6_PROXY_MASQ

SRv6 endpoint to SR-unaware appliance via masquerading proxy.

4.4.18 Encryption Tags

IPSECSW

Crypto in software.

IPSECHW

Crypto in hardware.

IPSECTRAN

IPSec in transport mode.

IPSECTUN

IPSec in tunnel mode.

IPSECINT

IPSec in interface mode.

AES

IPSec using AES algorithms.

AES_128_CBC

IPSec using AES 128 CBC algorithms.

AES_128_GCM

IPSec using AES 128 GCM algorithms.

AES_256_GCM

IPSec using AES 256 GCM algorithms.

HMAC

IPSec using HMAC integrity algorithms.

HMAC_SHA_256

IPSec using HMAC SHA 256 integrity algorithms.

HMAC_SHA_512

IPSec using HMAC SHA 512 integrity algorithms.

4.4.19 Client-Workload Tags

VM

All test cases which use at least one virtual machine.

LXC

All test cases which use Linux container and LXC utils.

DRC

All test cases which use at least one Docker container.

DOCKER

All test cases which use Docker as container manager.

APP

All test cases with specific APP use.

4.4.20 Container Orchestration Tags

K8S

All test cases which use Kubernetes for orchestration.

SFC_CONTROLLER

All test cases which use ligato/sfc_controller for driving configuration of vpp inside container.

VPP_AGENT

All test cases which use Golang implementation of a control/management plane for VPP

1VSWITCH

VPP running in Docker container acting as VSWITCH.

1VNF

1 VPP running in Docker container acting as VNF work load.

2VNF

2 VPP running in 2 Docker containers acting as VNF work load.

4VNF

4 VPP running in 4 Docker containers acting as VNF work load.

4.4.21 Multi-Threading Tags

STHREAD

Dynamic tag. All test cases using single poll mode thread.

MTHREAD

Dynamic tag. All test cases using more then one poll mode driver thread.

1NUMA

All test cases with packet processing on single socket.

2NUMA

All test cases with packet processing on two sockets.

1C

1 worker thread pinned to 1 dedicated physical core; or if HyperThreading is enabled, 2 worker threads each pinned to a separate logical core within 1 dedicated physical core. Main thread pinned to core 1.

2C

2 worker threads pinned to 2 dedicated physical cores; or if HyperThreading is enabled, 4 worker threads each pinned to a separate logical core within 2 dedicated physical cores. Main thread pinned to core 1.

4C

4 worker threads pinned to 4 dedicated physical cores; or if HyperThreading is enabled, 8 worker threads each pinned to a separate logical core within 4 dedicated physical cores. Main thread pinned to core 1.

1T1C

Dynamic tag. 1 worker thread pinned to 1 dedicated physical core. 1 receive queue per interface. Main thread pinned to core 1.

2T2C

Dynamic tag. 2 worker threads pinned to 2 dedicated physical cores. 1 receive queue per interface. Main thread pinned to core 1.

4T4C

Dynamic tag. 4 worker threads pinned to 4 dedicated physical cores. 2 receive queues per interface. Main thread pinned to core 1.

2T1C

Dynamic tag. 2 worker threads each pinned to a separate logical core within 1 dedicated physical core. 1 receive queue per interface. Main thread pinned to core 1.

4T2C

Dynamic tag. 4 worker threads each pinned to a separate logical core within 2 dedicated physical cores. 2 receive queues per interface. Main thread pinned to core 1.

8T4C

Dynamic tag. 8 worker threads each pinned to a separate logical core within 4 dedicated physical cores. 4 receive queues per interface. Main thread pinned to core 1.

4.4.22 Honeycomb Tags

HC_FUNC

Honeycomb functional test cases.

HC_NSH

Honeycomb NSH test cases.

HC_PERSIST

Honeycomb persistence test cases.

HC_REST_ONLY

(Exclusion tag) Honeycomb test cases that cannot be run in Netconf mode using ODL client for Restfconf -> Netconf translation.

BIBLIOGRAPHY

[lxc] Linux Containers[77]

[lxcnamespace] Resource management: Linux kernel Namespaces and cgroups[78].

[stgraber] LXC 1.0: Blog post series[79].

[lxcsecurity] Linux Containers Security[80].

[capabilities] Linux manual - capabilities - overview of Linux capabilities[81].

[cgroup1] Linux kernel documentation: cgroups[82].

[cgroup2] Linux kernel documentation: Control Group v2[83].

[selinux] SELinux Project Wiki[84].

[lxcsecfeatures] LXC 1.0: Security features[85].

[lxcsource] Linux Containers source[86].

[apparmor] Ubuntu AppArmor[87].

[seccomp] SECure COMPuting with filters[88].

[docker] Docker[89].

[k8sdoc] Kubernetes documentation[90].

[TWSLink] TWS[94]

[dockerhub] Docker hub[95]

[fdiocsitgerrit] FD.io/CSIT gerrit[96]

[fdioregistry] FD.io registy

[JenkinsSlaveDcrFile] jenkins-slave-dcr-file[97]

[77] https://linuxcontainers.org/
[78] https://www.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/lxc-namespace.pdf
[79] https://stgraber.org/2013/12/20/lxc-1-0-blog-post-series/
[80] https://linuxcontainers.org/lxc/security/
[81] http://man7.org/linux/man-pages/man7/capabilities.7.html
[82] https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt
[83] https://www.kernel.org/doc/Documentation/cgroup-v2.txt
[84] http://selinuxproject.org/page/Main_Page
[85] https://stgraber.org/2014/01/01/lxc-1-0-security-features/
[86] https://github.com/lxc/lxc
[87] https://wiki.ubuntu.com/AppArmor
[88] https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
[89] https://www.docker.com/what-docker
[90] https://kubernetes.io/docs/home/
[94] https://wiki.fd.io/view/CSIT/TWS
[95] https://hub.docker.com/
[96] https://gerrit.fd.io/r/CSIT
[97] https://github.com/snergfdio/multivppcache/blob/master/ubuntu18/Dockerfile

[CsitShimDcrFile] csit-shim-dcr-file[98]

[CsitSutDcrFile] csit-sut-dcr-file[99]

[ansiblelink] ansible[100]

[fdiocsitansible] Fd.io/CSIT ansible[101]

[inteli40e] Intel i40e[102]

[pciids] pci ids[103]

[98] https://github.com/snergfdio/multivppcache/blob/master/csit-shim/Dockerfile
[99] https://github.com/snergfdio/multivppcache/blob/master/csit-sut/Dockerfile
[100] https://www.ansible.com/
[101] https://git.fd.io/csit/tree/resources/tools/testbed-setup/ansible
[102] https://downloadmirror.intel.com/26370/eng/readme.txt
[103] http://pci-ids.ucw.cz/v2.2/pci.ids