
CSIT REPORT

Release rls1807

Aug 18, 2018

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Test Scenarios Overview	3
1.3	Physical Testbeds	4
1.4	Performance Test Methodology	9
2	VPP Performance	20
2.1	Overview	20
2.2	Release Notes	27
2.3	Packet Throughput	29
2.4	Speedup Multi-Core	206
2.5	Packet Latency	295
2.6	HTTP/TCP Performance	384
2.7	Comparisons	387
2.8	Throughput Trending	388
2.9	Test Environment	389
2.10	Documentation	402
3	DPDK Performance	410
3.1	Overview	410
3.2	Release Notes	412
3.3	Packet Throughput	414
3.4	Packet Latency	471
3.5	Comparisons	500
3.6	Throughput Trending	501
3.7	Test Environment	502
3.8	Documentation	513
4	VPP Functional	514
4.1	Overview	514
4.2	Release Notes	517
4.3	Test Environment	517
4.4	Documentation	527
5	HoneyComb Functional	528
5.1	Overview	528
5.2	Release Notes	531
5.3	Test Environment	532
5.4	Documentation	532
6	NSH_SFC Functional	533

6.1	Overview	533
6.2	Release Notes	534
6.3	Test Environment	534
6.4	Documentation	534
7	DMM Functional	535
7.1	Overview	535
7.2	Release Notes	536
7.3	Test Environment	536
7.4	Documentation	536
8	CSIT Framework Documentation	537
8.1	Design	537
8.2	Test Naming	541
8.3	Presentation and Analytics Layer	543
8.4	CSIT RF Tags Descriptions	570
	Bibliography	582

1.1 Introduction

FD.io CSIT-18.07 report contains system performance and functional testing data of VPP-18.07 release. [PDF version of this report](#)¹ is available for download.

CSIT-18.07 report is structured as follows:

1. INTRODUCTION: General introduction to FD.io CSIT-18.07.
 - **Introduction:** This section.
 - **Test Scenarios Overview:** A brief overview of test scenarios covered in this report.
 - **Physical Testbeds:** Description of physical testbeds.
 - **Performance Test Methodology:** Benchmarking 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-18.07, 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 repeatability of measurements.
 - **Packet Latency:** Latency graphs based on results from test job executions.
 - **Speedup Multi-Core:** NDR, PDR throughput multi-core speedup graphs based on results from test job executions.
 - **HTTP/TCP Performance:** HTTP/TCP VPP test server performance graphs.
 - **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.

¹ https://docs.fd.io/csit/rls1807/report/_static/archive/csit_rls1807.pdf

- **Documentation:** Documentation of K8s Pod/Container orchestration in CSIT and 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-18.07, known issues.
 - **Packet Throughput:** NDR, PDR throughput graphs based on results from repeated same test job executions to verify repeatability 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 FUNCTIONAL: VPP functional tests executed in virtual FD.io testbeds.
 - **Overview:** Tested virtual topologies, test coverage and naming specifics;
 - **Release Notes:** Changes in CSIT-18.07, added tests, environment or methodology changes, known issues.
 - **Test Environment:** Functional test environment configuration.
 - **Documentation:** Pointers to CSIT source code documentation for VPP functional tests.
 5. HONEYCOMBE FUNCTIONAL: Honeycomb functional tests executed in virtual FD.io testbeds.
 - **Overview:** Tested virtual topologies, test coverage and naming specifics;
 - **Release Notes:** Changes in CSIT-18.07, known issues.
 - **Test Environment:** Functional test environment configuration.
 - **Documentation:** Pointers to CSIT source code documentation for Honeycomb functional tests.
 6. NSH_SFC FUNCTIONAL: NSH_SFC functional tests executed in virtual FD.io testbeds.
 - **Overview:** Tested virtual topologies, test coverage and naming specifics;
 - **Release Notes:** Changes in CSIT-18.07, known issues.
 - **Test Environment:** Functional test environment configuration.
 - **Documentation:** Pointers to CSIT source code documentation for NSH_SFC functional tests.
 7. DMM FUNCTIONAL: DMM functional tests executed in virtual FD.io testbeds.
 - **Overview:** Tested virtual topologies, test coverage and naming specifics;
 - **Release Notes:** Changes in CSIT-18.07, known issues.
 - **Test Environment:** Functional test environment configuration.
 - **Documentation:** Pointers to CSIT source code documentation for DMM functional tests.
 8. 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.
 - **VPP K8s Container Memif:** VPP K8s Container/Pod topologies NDR/PDR throughput.
 - **DPDK Performance:** DPDK Testpmd and L3fwd NDR/PDR throughput and latency.
 - **VPP Functional:** Detailed VPP functional results.

- **Honeycomb Functional:** Detailed HoneyComb functional results.
 - **NSH_SFC Functional:** Detailed nsh-plugin functional results.
 - **DMM Functional:** Detailed DMM functional results.
9. 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.
 - **VPP K8s Container Memif:** Configuration data.
 - **VPP Functional:** Configuration data.
 10. 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.
 11. 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.

1.2 Test Scenarios Overview

FD.io CSIT-18.07 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, 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 Testpmd-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 DPDK Testpmd 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. **K8s Container/Pod Topologies Performance:** VPP container performance tests using memif for interconnecting VPP-in- Container/Pod instances orchestrated by K8s integrated with [Ligato](https://github.com/ligato)² for container networking. TRex is used as a traffic generator.

² <https://github.com/ligato>

5. **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.
6. **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.
7. **Honeycomb Functional:** Honeycomb functional tests are executed in virtual FD.io testbeds, focusing on Honeycomb management and programming functionality of VPP. Tests cover a range of CRUD operations executed against VPP.
8. **NSH_SFC Functional:** NSH_SFC functional tests are executed in virtual FD.io testbeds focusing on VPP nsh-plugin data plane functionality. Scapy is used as a traffic generator.
9. **DMM Functional:** DMM functional tests are executed in virtual FD.io testbeds demonstrating a single server (DUT1) and single client (DUT2) scenario using DMM framework and Linux kernel TCP/IP stack.

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-18.07 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-18.07 source code for the executed test suites is available in CSIT branch rls1807 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`.

1.3 Physical Testbeds

All FD.io (Fast Data Input/Output) CSIT (Continuous System Integration and Testing) performance testing listed in this report are executed on physical testbeds built with bare-metal servers hosted by LF FD.io project. Two testbed topologies 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. Used for executing all of the data plane tests including overlay tunnels and IPSec tests.
- **2-Node Topology:** Consisting of one server acting as SUTs (Systems Under Test) and one server as TG (Traffic Generator), both connected in ring topology. Used for execution of tests without any overlay tunnel encapsulations. Added in CSIT rls18.07.

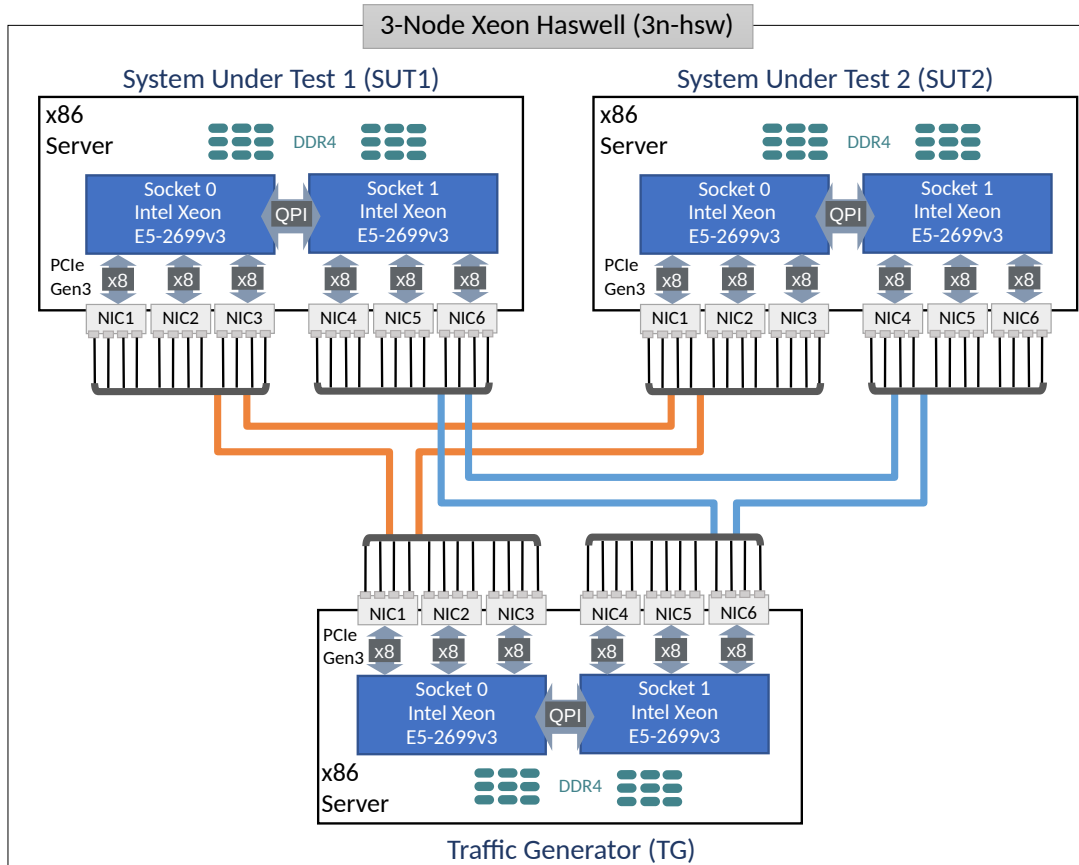
Current FD.io production testbeds are built with servers based on two processor generations of Intel Xeons: Haswell-SP (E5-2699v3) and Skylake (Platinum 8180). Testbeds built with servers based on Arm processors are in the process of being added to FD.io production.

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

Following sections describe existing production testbed types.

1.3.1 3-Node Xeon Haswell (3n-hsw)

3n-hsw testbed is based on three Cisco UCS-c240m3 servers each equipped with two Intel Xeon Haswell-SP E5-2699v3 2.3 GHz 18 core processors. Physical testbed topology is depicted in a figure below.



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 x1710-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 x1710-QDA2 2p40GE Intel.
2. NIC-2: NIC x710-DA2 2p10GE Intel.
3. NIC-3: empty.
4. NIC-4: NIC x1710-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.

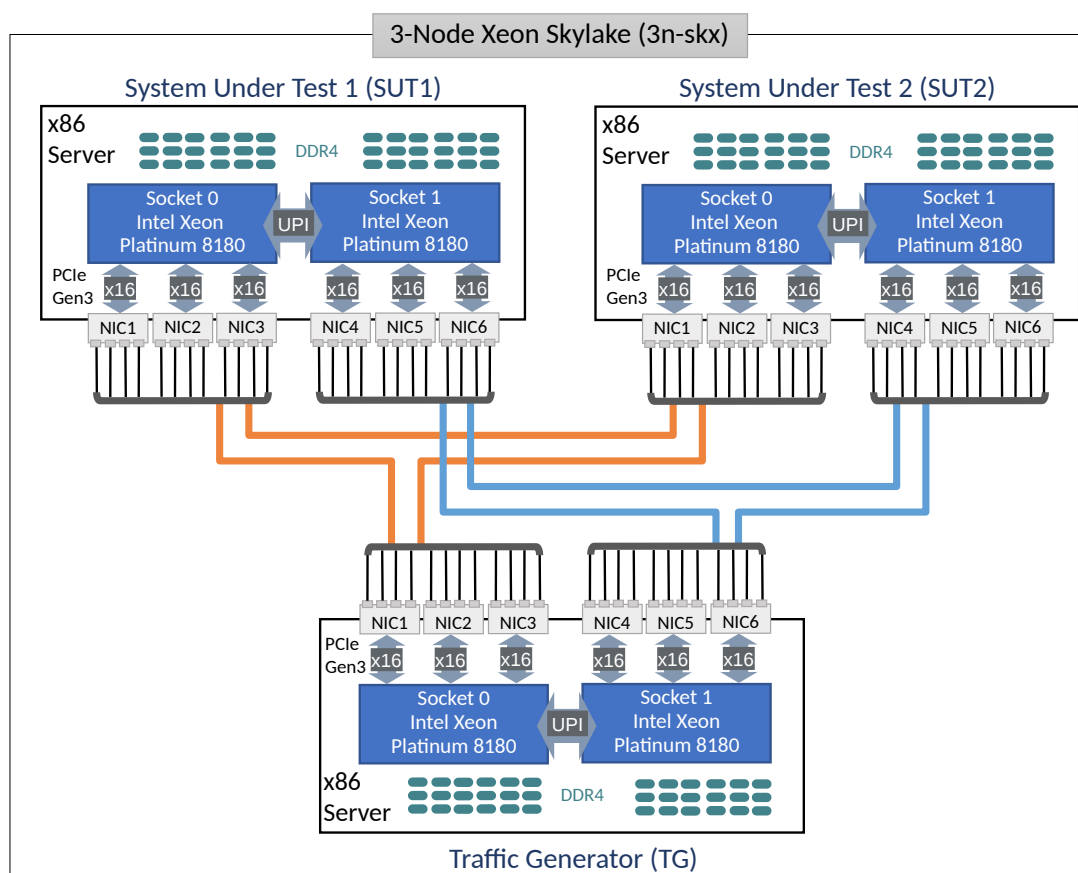
Complete 3n-hsw testbeds specification is available on [CSIT LF testbed³](https://wiki.fd.io/view/CSIT/CSIT_LF_testbed) wiki page.

Total of three 3n-hsw testbeds are in operation in FD.io labs.

³ https://wiki.fd.io/view/CSIT/CSIT_LF_testbed

1.3.2 3-Node Xeon Skylake (3n-skx)

3n-skx testbed is based on three SuperMicro SYS-7049GP-TRT servers each equipped with two Intel Xeon Skylake Platinum 8180 2.5 GHz 28 core processors. Physical testbed topology is depicted in a figure below.



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.

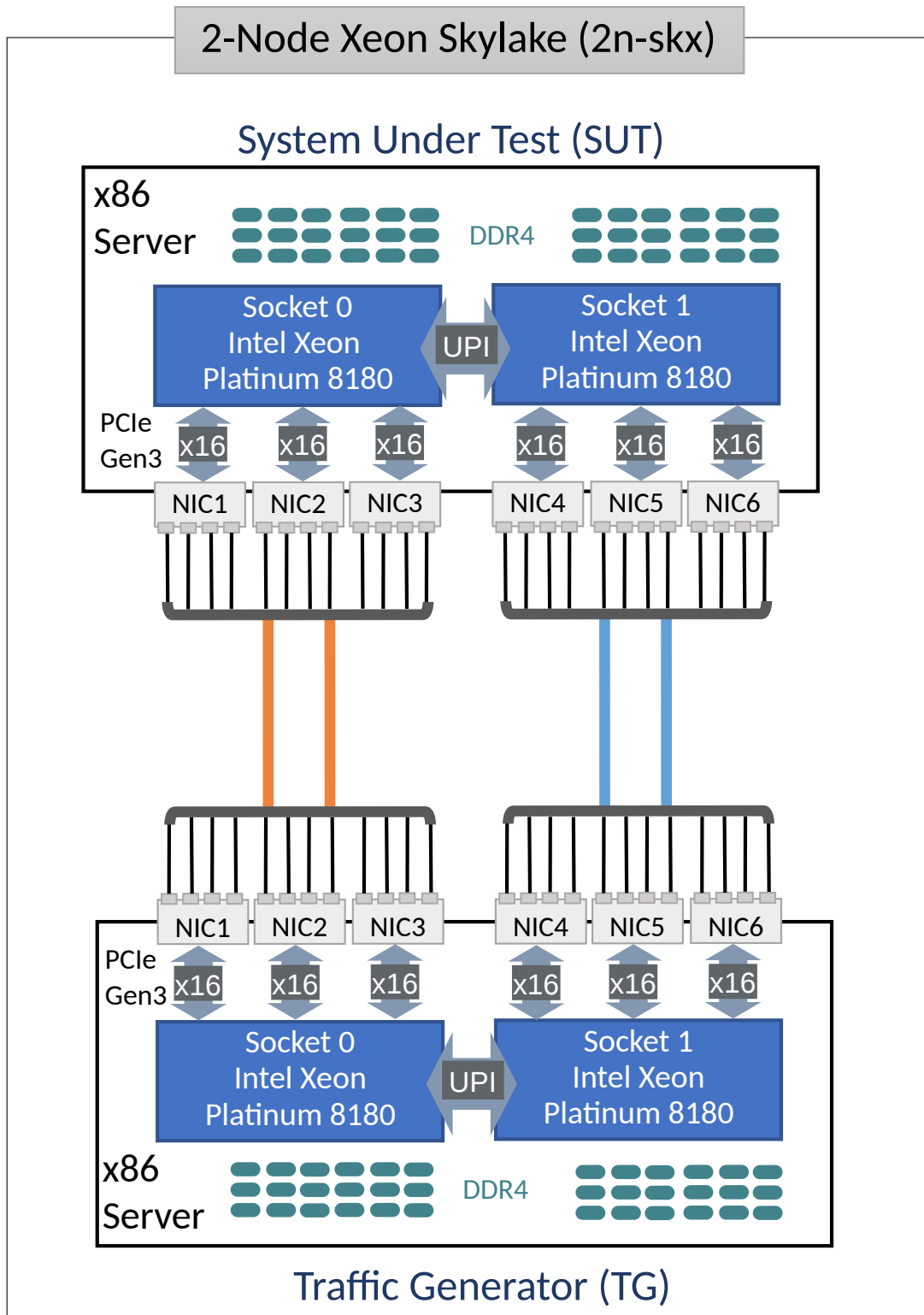
Complete 3n-skx testbeds specification is available on [CSIT LF lab extension](#)⁴ wiki page.

Total of two 3n-skx testbeds are in operation in FD.io labs.

1.3.3 2-Node Xeon Skylake (2n-skx)

2n-skx testbed is based on two SuperMicro SYS-7049GP-TRT servers each equipped with two Intel Xeon Skylake Platinum 8180 2.5 GHz 28 core processors. Physical testbed topology is depicted in a figure below.

⁴ https://wiki.fd.io/view/CSIT/fdio_csit_lab_ext_llid_draft



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: mcx556a-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: mcx556a-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.

Complete 2n-skx testbed specification is available on [CSIT/Testbeds: Xeon Skx, Arm, Atom](#)⁵ wiki page.

Total of four 2n-skx testbeds are in operation in FD.io labs.

1.4 Performance Test Methodology

1.4.1 Throughput

Packet and bandwidth throughput are measured in accordance with [RFC 2544](#)⁶, using FD.io CSIT Multiple Loss Ratio search (MLRsearch), an optimized binary search algorithm, that measures SUT/DUT throughput at different Packet Loss Ratio (PLR) values.

Following MLRsearch values are measured across a range of L2 frame sizes and reported:

- NON DROP RATE (NDR): packet and bandwidth throughput at PLR=0%.
 - **Aggregate packet rate:** NDR_LOWER <bi-directional packet rate> pps.
 - **Aggregate bandwidth rate:** NDR_LOWER <bi-directional bandwidth rate> Gbps.
- PARTIAL DROP RATE (PDR): packet and bandwidth throughput at PLR=0.5%.
 - **Aggregate packet rate:** PDR_LOWER <bi-directional packet rate> pps.
 - **Aggregate bandwidth rate:** PDR_LOWER <bi-directional bandwidth rate> Gbps.

NDR and PDR are measured for the following L2 frame sizes (untagged Ethernet):

- IPv4 payload: 64B, IMIX_v4_1 (28x64B, 16x570B, 4x1518B), 1518B, 9000B.
- IPv6 payload: 78B, 1518B, 9000B.

All rates are reported from external Traffic Generator perspective.

1.4.2 MLRsearch Algorithm

Multiple Loss Rate search (MLRsearch) is a 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 distinct 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 execution time compared to separate binary searches for NDR and PDR. MLRsearch reduces execution time even further by relying on shorter trial durations of intermediate steps, with only the final measurements conducted at the specified

⁵ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.

⁶ <https://tools.ietf.org/html/rfc2544.html>

final trial duration. This results in the shorter overall search execution time when compared to a standard NDR/PDR binary search, while guaranteeing the same or similar results.

If needed, 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.

Overview

The main properties of MLRsearch:

- MLRsearch is a duration aware multi-phase multi-rate search algorithm.
 - Initial phase determines promising starting interval for the search.
 - Intermediate phases progress towards defined final search criteria.
 - Final phase executes measurements according to the final search criteria.
- *Initial phase:*
 - Uses link rate as a starting transmit rate and discovers the Maximum Receive Rate (MRR) used as an input to the first intermediate phase.
- *Intermediate phases:*
 - Start with initial trial duration (in the first phase) and converge geometrically towards the final trial duration (in the final phase).
 - Track two values for NDR and two for PDR.
 - * The values are called (NDR or PDR) lower_bound and upper_bound.
 - * Each value comes from a specific trial measurement (most recent for that transmit rate), and as such the value is associated with that measurement's duration and loss.
 - * A bound can be invalid, for example if NDR lower_bound has been measured with nonzero loss.
 - * Invalid bounds are not real boundaries for the searched value, but are needed to track interval widths.
 - * Valid bounds are real boundaries for the searched value.
 - * Each non-initial phase ends with all bounds valid.
 - Start with a large (lower_bound, upper_bound) interval width and geometrically converge towards the width goal (measurement resolution) of the phase. Each phase halves the previous width goal.
 - Use internal and external searches:
 - * External search - measures at transmit rates outside the (lower_bound, upper_bound) interval. Activated when a bound is invalid, to search for a new valid bound by doubling the interval width. It is a variant of [exponential search](https://en.wikipedia.org/wiki/Exponential_search)⁷.
 - * Internal search - [binary search](https://en.wikipedia.org/wiki/Binary_search)⁸, measures at transmit rates within the (lower_bound, upper_bound) valid interval, halving the interval width.
- *Final phase* is executed with the final test trial duration, and the final width goal that determines resolution of the overall search. Intermediate phases together with the final phase are called non-initial phases.

⁷ https://en.wikipedia.org/wiki/Exponential_search

⁸ https://en.wikipedia.org/wiki/Binary_search

The main benefits of MLRsearch vs. binary search include:

- In general MLRsearch is likely to execute more search trials overall, but less trials at a set final duration.
- In well behaving cases it greatly reduces (>50%) the overall duration compared to a single PDR (or NDR) binary search duration, while finding multiple drop rates.
- In all cases MLRsearch yields the same or similar results to binary search.
- Note: both binary search and MLRsearch are susceptible to reporting non-repeatable results across multiple runs for very bad behaving cases.

Caveats:

- Worst case MLRsearch can take longer than a binary search e.g. in case of drastic changes in behaviour for trials at varying durations.

Search Implementation

Following is a brief description of the current MLRsearch implementation in FD.io CSIT.

Input Parameters

1. *maximum_transmit_rate* - maximum packet transmit rate to be used by external traffic generator, limited by either the actual Ethernet link rate or traffic generator NIC model capabilities. Sample defaults: 2 * 14.88 Mpps for 64B 10GE link rate, 2 * 18.75 Mpps for 64B 40GE NIC maximum rate.
2. *minimum_transmit_rate* - minimum packet transmit rate to be used for measurements. MLRsearch fails if lower transmit rate needs to be used to meet search criteria. Default: 2 * 10 kpps (could be higher).
3. *final_trial_duration* - required trial duration for final rate measurements. Default: 30 sec.
4. *initial_trial_duration* - trial duration for initial MLRsearch phase. Default: 1 sec.
5. *final_relative_width* - required measurement resolution expressed as (lower_bound, upper_bound) interval width relative to upper_bound. Default: 0.5%.
6. *packet_loss_ratio* - maximum acceptable PLR search criteria for PDR measurements. Default: 0.5%.
7. *number_of_intermediate_phases* - number of phases between the initial phase and the final phase. Impacts the overall MLRsearch duration. Less phases are required for well behaving cases, more phases may be needed to reduce the overall search duration for worse behaving cases. Default (2). (Value chosen based on limited experimentation to date. More experimentation needed to arrive to clearer guidelines.)

Initial phase

1. First trial measures at maximum rate and discovers MRR.
 - (a) *in*: trial_duration = initial_trial_duration.
 - (b) *in*: offered_transmit_rate = maximum_transmit_rate.
 - (c) *do*: single trial.
 - (d) *out*: measured loss ratio.
 - (e) *out*: mrr = measured receive rate.
2. Second trial measures at MRR and discovers MRR2.
 - (a) *in*: trial_duration = initial_trial_duration.

- (b) *in*: offered_transmit_rate = MRR.
 - (c) *do*: single trial.
 - (d) *out*: measured loss ratio.
 - (e) *out*: mrr2 = measured receive rate.
3. Third trial measures at MRR2.
- (a) *in*: trial_duration = initial_trial_duration.
 - (b) *in*: offered_transmit_rate = MRR2.
 - (c) *do*: single trial.
 - (d) *out*: measured loss ratio.

Non-initial phases

1. Main loop:

- (a) *in*: trial_duration for the current phase. Set to initial_trial_duration for the first intermediate phase; to final_trial_duration for the final phase; or to the element of interpolating geometric sequence for other intermediate phases. For example with two intermediate phases, trial_duration of the second intermediate phase is the geometric average of initial_trial_duration and final_trial_duration.
- (b) *in*: relative_width_goal for the current phase. Set to final_relative_width for the final phase; doubled for each preceding phase. For example with two intermediate phases, the first intermediate phase uses quadruple of final_relative_width and the second intermediate phase uses double of final_relative_width.
- (c) *in*: ndr_interval, pdr_interval from the previous main loop iteration or the previous phase. If the previous phase is the initial phase, both intervals have lower_bound = MRR2, upper_bound = MRR. Note that the initial phase is likely to create intervals with invalid bounds.
- (d) *do*: According to the procedure described in point 2, either exit the phase (by jumping to 1.g.), or prepare new transmit rate to measure with.
- (e) *do*: Perform the trial measurement at the new transmit rate and trial_duration, compute its loss ratio.
- (f) *do*: Update the bounds of both intervals, based on the new measurement. The actual update rules are numerous, as NDR external search can affect PDR interval and vice versa, but the result agrees with rules of both internal and external search. For example, any new measurement below an invalid lower_bound becomes the new lower_bound, while the old measurement (previously acting as the invalid lower_bound) becomes a new and valid upper_bound. Go to next iteration (1.c.), taking the updated intervals as new input.
- (g) *out*: current ndr_interval and pdr_interval. In the final phase this is also considered to be the result of the whole search. For other phases, the next phase loop is started with the current results as an input.

2. New transmit rate (or exit) calculation (for 1.d.):

- If there is an invalid bound then prepare for external search:
 - If the most recent measurement at NDR lower_bound transmit rate had the loss higher than zero, then the new transmit rate is NDR lower_bound decreased by two NDR interval widths.
 - Else, if the most recent measurement at PDR lower_bound transmit rate had the loss higher than PLR, then the new transmit rate is PDR lower_bound decreased by two PDR interval widths.

- Else, if the most recent measurement at NDR upper_bound transmit rate had no loss, then the new transmit rate is NDR upper_bound increased by two NDR interval widths.
- Else, if the most recent measurement at PDR upper_bound transmit rate had the loss lower or equal to PLR, then the new transmit rate is PDR upper_bound increased by two PDR interval widths.
- If interval width is higher than the current phase goal:
 - Else, if NDR interval does not meet the current phase width goal, prepare for internal search. The new transmit rate is $(\text{NDR lower bound} + \text{NDR upper bound}) / 2$.
 - Else, if PDR interval does not meet the current phase width goal, prepare for internal search. The new transmit rate is $(\text{PDR lower bound} + \text{PDR upper bound}) / 2$.
- Else, if some bound has still only been measured at a lower duration, prepare to re-measure at the current duration (and the same transmit rate). The order of priorities is:
 - NDR lower_bound,
 - PDR lower_bound,
 - NDR upper_bound,
 - PDR upper_bound.
- Else, do not prepare any new rate, to exit the phase. This ensures that at the end of each non-initial phase all intervals are valid, narrow enough, and measured at current phase trial duration.

Implementation Deviations

This document so far has been describing a simplified version of MLRsearch algorithm. The full algorithm as implemented contains additional logic, which makes some of the details (but not general ideas) above incorrect. Here is a short description of the additional logic as a list of principles, explaining their main differences from (or additions to) the simplified description, but without detailing their mutual interaction.

1. *Logarithmic transmit rate.* In order to better fit the relative width goal, the interval doubling and halving is done differently. For example, the middle of 2 and 8 is 4, not 5.
2. *Optimistic maximum rate.* The increased rate is never higher than the maximum rate. Upper bound at that rate is always considered valid.
3. *Pessimistic minimum rate.* The decreased rate is never lower than the minimum rate. If a lower bound at that rate is invalid, a phase stops refining the interval further (until it gets re-measured).
4. *Conservative interval updates.* Measurements above current upper bound never update a valid upper bound, even if drop ratio is low. Measurements below current lower bound always update any lower bound if drop ratio is high.
5. *Ensure sufficient interval width.* Narrow intervals make external search take more time to find a valid bound. If the new transmit increased or decreased rate would result in width less than the current goal, increase/decrease more. This can happen if the measurement for the other interval makes the current interval too narrow. Similarly, take care the measurements in the initial phase create wide enough interval.
6. *Timeout for bad cases.* The worst case for MLRsearch is when each phase converges to intervals way different than the results of the previous phase. Rather than suffer total search time several times larger than pure binary search, the implemented tests fail themselves when the search takes too long (given by argument *timeout*).

1.4.3 Maximum Receive Rate MRR

MRR tests measure the 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.

Current parameters for MRR tests:

- Ethernet frame sizes: 64B (78B for IPv6), IMIX, 1518B, 9000B; all quoted sizes include frame CRC, but exclude per frame transmission overhead of 20B (preamble, inter frame gap).
- Maximum load offered: 10GE and 40GE link (sub-)rates depending on NIC tested, with the actual packet rate depending on frame size, transmission overhead and traffic generator NIC forwarding capacity.
 - For 10GE NICs the maximum packet rate load is 2* 14.88 Mpps for 64B, a 10GE bi-directional link rate.
 - For 25GE NICs the maximum packet rate load is 2* 18.75 Mpps for 64B, a 25GE bi-directional link sub-rate limited by TG 25GE NIC used, XXV710.
 - For 40GE NICs the maximum packet rate load is 2* 18.75 Mpps for 64B, a 40GE bi-directional link sub-rate limited by TG 40GE NIC used, XL710. Packet rate for other tested frame sizes is limited by PCIe Gen3 x8 bandwidth limitation of ~50Gbps.
- Trial duration: 10sec.

Similarly to NDR/PDR throughput tests, MRR test should be reporting bi-directional link rate (or NIC rate, if lower) if tested VPP configuration can handle the packet rate higher than bi-directional link rate, e.g. large packet tests and/or multi-core tests.

MRR tests are used for continuous performance trending and for comparison between releases. Daily trending job tests subset of frame sizes, focusing on 64B (78B for IPv6) for all tests and IMIX for selected tests (vhost, memif).

1.4.4 Packet Latency

TRex Traffic Generator (TG) is used for measuring latency of VPP DUTs. Reported latency values are measured using following methodology:

- Latency tests are performed at 100% of discovered NDR and PDR rates for each throughput test and packet size (except IMIX).
- 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 latency values per stream direction, hence two sets of latency values are reported per test case; future release of TRex is expected to report latency percentiles.
- Reported latency values are aggregate across two SUTs due to three node topology used for all performance tests; for per SUT latency, reported value should be divided by two.
- 1usec is the measurement accuracy advertised by TRex TG for the setup used in FD.io labs used by CSIT project.
- TRex setup introduces an always-on error of about 2*2usec per latency flow additional Tx/Rx interface latency induced by TRex SW writing and reading packet timestamps on CPU cores without HW acceleration on NICs closer to the interface line.

1.4.5 Multi-Core Speedup

All performance tests are executed with single processor 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-18.07 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 testbeds.

More information about physical testbeds is provided in *Physical Testbeds* (page 4).

Multi-core Tests

CSIT-18.07 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.
 2. 2t2c - 2 VPP worker threads on 2 physical cores.
 3. 4t4c - 4 VPP worker threads on 4 physical cores.
1. Intel Xeon Skylake testbeds (2n-skx, 3n-skx) 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 throughput 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.

1.4.6 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](#)⁹ 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.

⁹ <https://git.fd.io/vpp/tree/src/vpp/conf/startup.conf?h=stable/1807>

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. socket-mem <value>,<value> - memory per numa. (Not required anymore due to VPP code changes, should be removed in CSIT-18.10.)

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. num-rx-desc/num-tx-desc - number of rx/tx descriptors for specific NICs, incl. xl710, x710, xxv710.
4. num-mbufs <value> - increases number of buffers allocated, needed only in scenarios with large number of interfaces and worker threads. Value is per CPU socket. Default is 16384.
5. 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.
6. UIO driver - depends on topology file definition.
7. QAT VFs - depends on NRThreads, each thread = 1QAT VFs.

1.4.7 KVM VMs vhost-user

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

- Tests with varying Qemu virtio queue (a.k.a. vring) sizes: [vr256] default 256 descriptors, [vr1024] 1024 descriptors to optimize for packet throughput.
- Tests with varying Linux CFS (Completely Fair Scheduler) settings: [cfs] default settings, [cfsrr1] CFS RoundRobin(1) policy applied to all data plane threads handling test packet path including all VPP worker threads and all Qemu testpmd poll-mode threads.
- Resulting test cases are all combinations with [vr256,vr1024] and [cfs,cfsrr1] settings.
- Adjusted Linux kernel CFS scheduler policy for data plane threads used in CSIT is documented in [CSIT Performance Environment Tuning wiki](#)¹⁰.
- The purpose is to verify performance impact (MRR and NDR/PDR throughput) and same test measurements repeatability, by making VPP and VM data plane threads less susceptible to other Linux OS system tasks hijacking CPU cores running those data plane threads.

1.4.8 LXC/DRC Container Memif

CSIT-18.07 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). LXC and DRCs run in a privileged 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 20).

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.

¹⁰ <https://wiki.fd.io/view/CSIT/csit-perf-env-tuning-ubuntu1604>

More information about CSIT LXC and DRC setup and control is available in *Container Orchestration in CSIT* (page 402).

1.4.9 K8s Container Memif

CSIT-18.07 includes tests of VPP topologies running in K8s orchestrated Pods/Containers and connected over memif virtual interfaces. In order to provide simple topology coding flexibility and extensibility container orchestration is done with [Kubernetes](#)¹¹ using [Docker](#)¹² images for all container applications including VPP. [Ligato](#)¹³ is used for the Pod/Container networking orchestration that is integrated with K8s, including memif support.

In these tests VPP vswitch runs in a K8s Pod with Docker Container (DRC) handling NIC interfaces and connecting over memif to more instances of VPP running in Pods/DRCs. All DRCs run in a privileged 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 *Physical Testbeds* (page 4).

Further documentation is available in *Container Orchestration in CSIT* (page 402).

1.4.10 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-18.07 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.

1.4.11 TRex Traffic Generator

Usage

[TRex traffic generator](#)¹⁴ is used for all CSIT performance tests. TRex stateless mode is used to measure NDR and PDR throughputs using binary search (NDR and PDR discovery tests) and for quick checks of DUT performance against the reference NDRs (NDR check tests) for specific configuration.

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](#)¹⁵.
- 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 -c 7 --iom 0 > /tmp/trex.  
↪ log 2>&1 &' > /dev/null
```

¹¹ <https://github.com/kubernetes>

¹² <https://github.com/docker>

¹³ <https://github.com/ligato>

¹⁴ <https://wiki.fd.io/view/TRex>

¹⁵ https://git.fd.io/csit/tree/resources/tools/trex/trex_installer.sh?h=rls1807

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

1.4.12 HTTP/TCP with WRK tool

WRK HTTP benchmarking tool¹⁶ is used for experimental TCP/IP and HTTP tests of VPP TCP/IP 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 quite high scale benchmarking of the main TCP/IP and HTTP service including HTTP TCP/IP Connections-Per-Second (CPS), HTTP Requests-Per-Second and HTTP Bandwidth Throughput.

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.
 - Timeout for server responses set to 5 seconds.
 - Test duration is 30 seconds.

¹⁶ <https://github.com/wg/wrk>

- 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.

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 4).

2.1.1 Logical Topologies

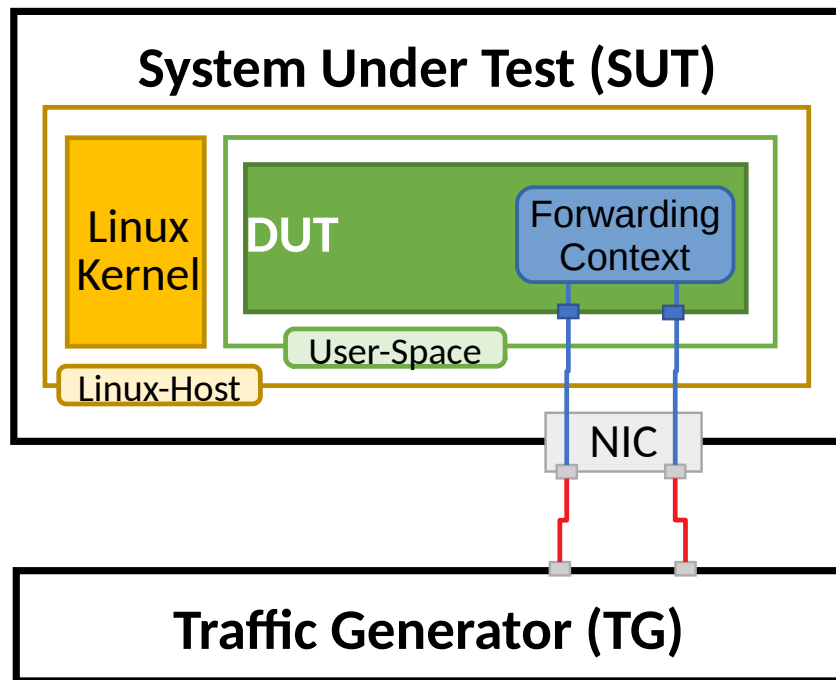
CSIT VPP performance tests are executed on physical testbeds described in *Physical Testbeds* (page 4). Based on the packet path thru server SUTs, three distinct logical topology types are used for VPP DUT data plane testing:

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

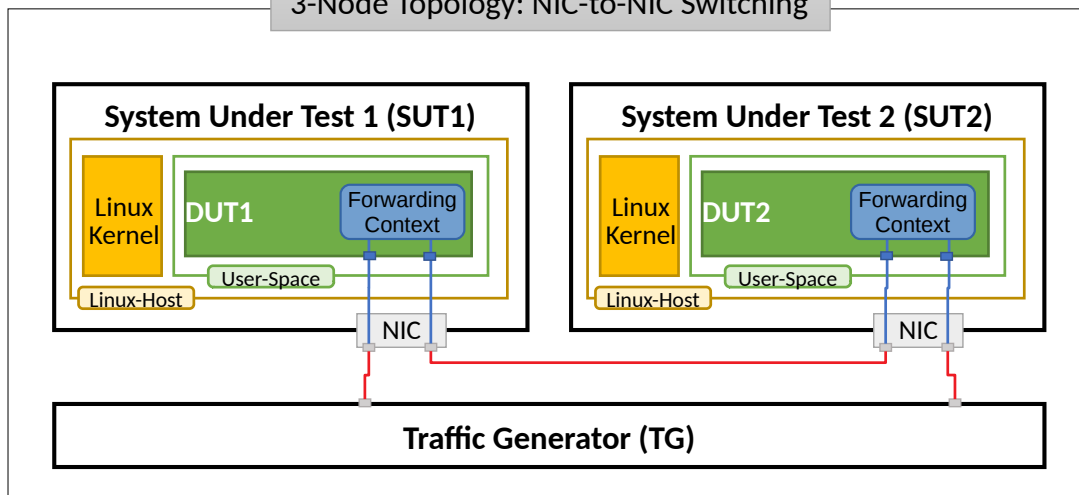
NIC-to-NIC Switching

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.

2-Node Topology: NIC-to-NIC Switching



3-Node Topology: NIC-to-NIC Switching



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 limitations. 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 throughput 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.

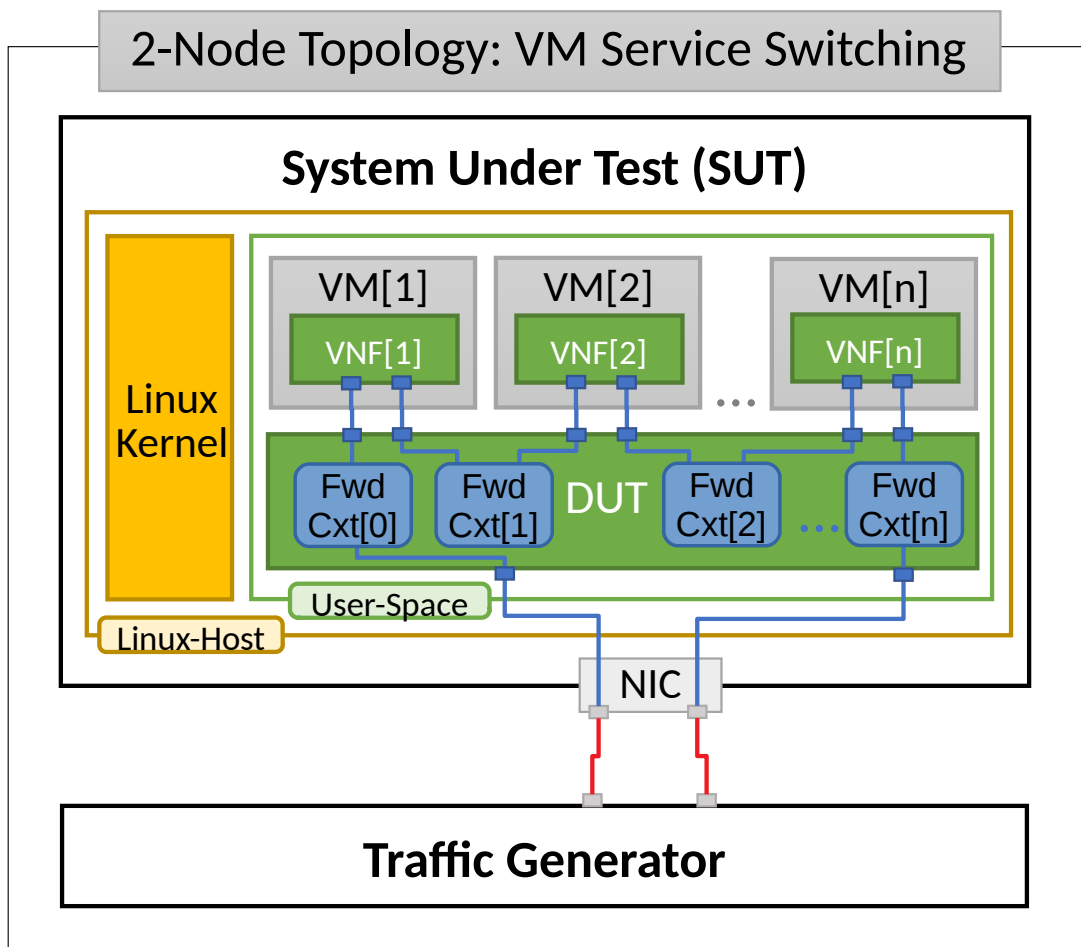
VM Service Switching

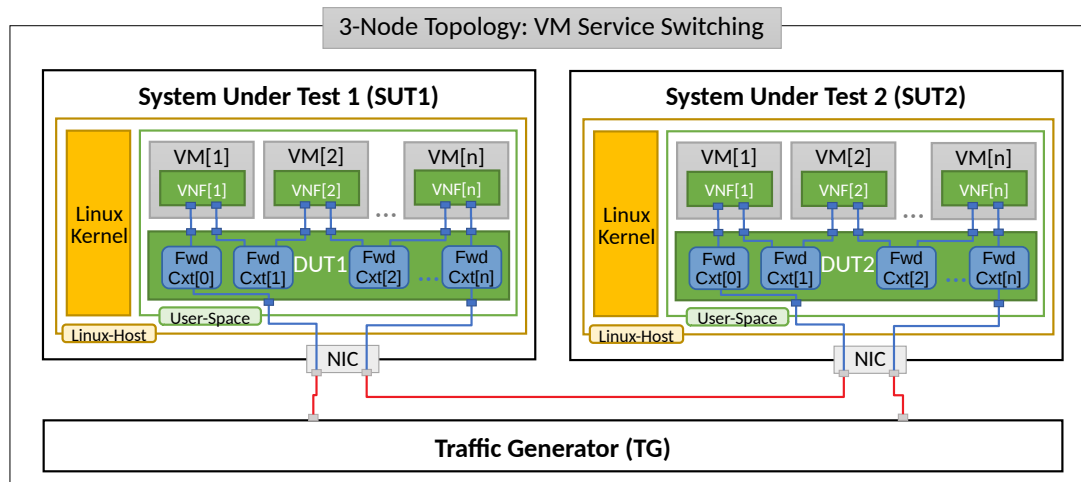
VM service switching topology test cases require VPP DUT to communicate with Virtual Machines (VMs) over vhost-user virtual interfaces.

Two types of VM service topologies are tested in CSIT-18.07:

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

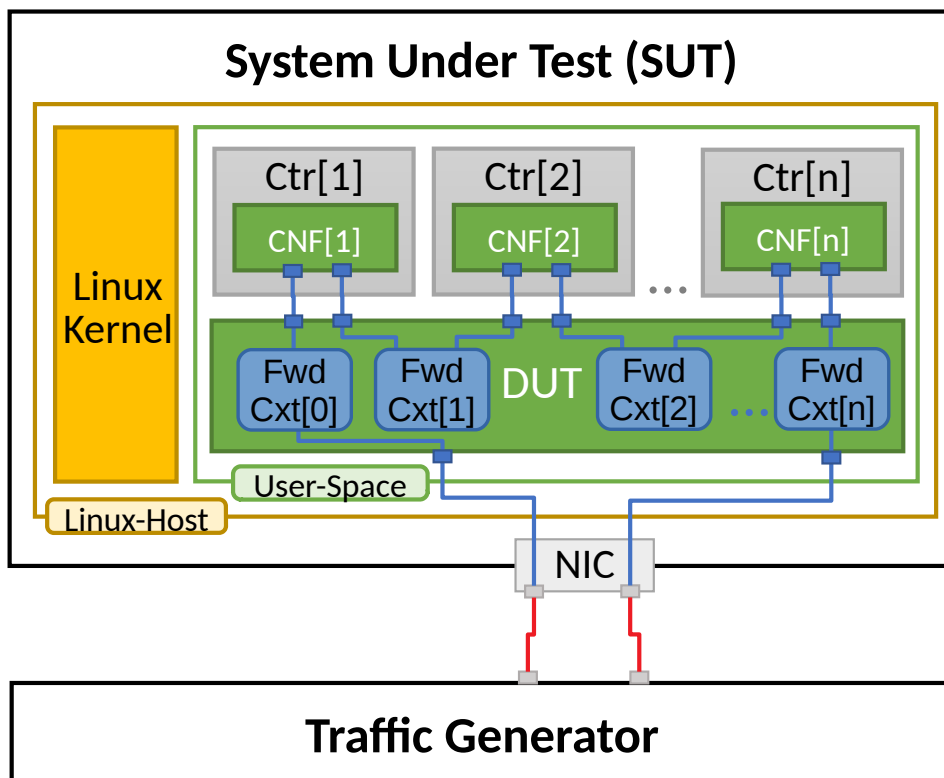
Container service switching topology test cases require VPP DUT to communicate with Containers (Ctrs) over memif virtual interfaces.

Three types of VM service topologies are tested in CSIT-18.07:

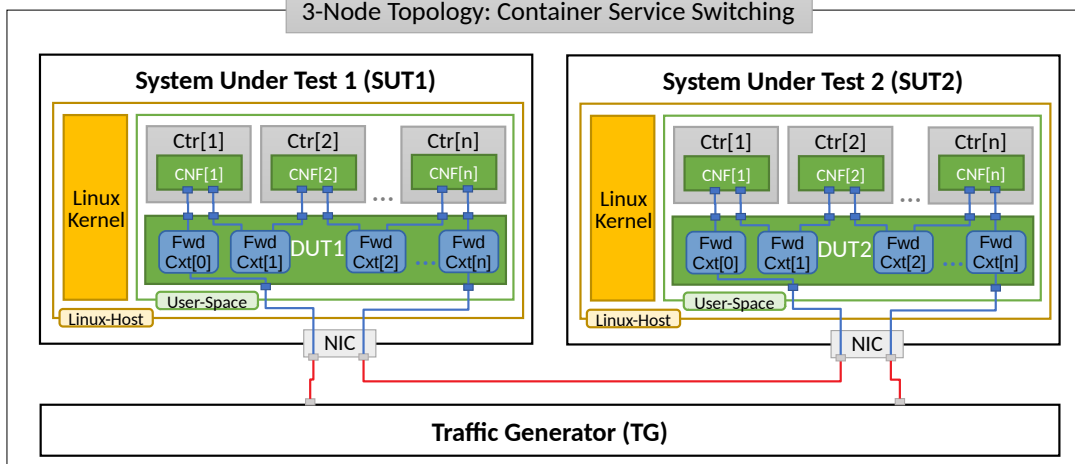
1. “Parallel” topology with packets flowing within SUT from NIC(s) via VPP DUT to Container, back to VPP DUT, then out thru NIC(s).
2. “Chained” topology (a.k.a. “Snake”) with packets flowing within SUT from NIC(s) via VPP DUT to Container, back to VPP DUT, then to the next Container, back to VPP DUT 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.

2-Node Topology: Container Service Switching



3-Node Topology: Container Service Switching



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**¹⁷, 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-18.07 includes following VPP data plane functionality performance tested across a range of NIC drivers and NIC models:

¹⁷ <https://tools.ietf.org/html/rfc2544.html>

Functionality	Description
ACL	L2 Bridge-Domain switching and IPv4 and 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-SHA1 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-SHA1 ciphers, in combination with IPv4 routing.
K8s Containers Memif	K8s orchestrated container VPP service chain topologies connected over the memif virtual interface.
KVM VMs vhost-user	Virtual topologies with service chains of 1 and 2 VMs 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-18.07 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 541).

2.2 Release Notes

2.2.1 Changes in CSIT-18.07

1. VPP PERFORMANCE TESTS

- **MRR Throughput:** MRR (Maximum Receive Rate) test code has now configurable trial duration and number of consecutive executions. Coverage of MRR tests has been extended across more test scenarios. MRR tests are used for continuous performance trending and for comparison between VPP releases.
- **MLRsearch Throughput:** MLRsearch algorithm has been introduced for all NDR and PDR throughput tests. All tests that previously used binary search got converted to MLRsearch. Coverage of NDR/PDR tests has been extended across more test scenarios.
- **L2patch Tests:** Tests measure performance of VPP L2patch, the fastest L2 forwarding path implemented in VPP, that cross-links RX and TX of two physical interfaces.
- **2-Node Tests:** A new baseline set of 2-node tests covering base ip4, ip6, l2patch, l2bd, l2xc, running on new Xeon Skylake testbeds.
- **Generated tests:** Simplified and unified test structure, semi- autogenerated by generator script. Test generator is currently able to create test combinations with various frame size and cores combinations. All existing test cases were converted to new format.
- **Simultaneous Multi-Threading:** SMT-aware detection of server processor operation mode (HyperThreading enabled/disabled) with associated compute resource configuration including thread affinity, number of Rx queues and DPDK I/O mbufs. Tests are automatically tagged during execution to indicate executed thread configuration.
- **Intel Xeon Skylake Support:** Support for 2-Node and 3-Node physical testbed topologies based on the new SuperMirco servers each with two Intel Xeon Skylake Platinum processors. Full Ansible playbooks refactor for quick server (re)installation and reference pointers of configuration.

2. PRESENTATION AND ANALYTICS LAYER

- **Performance trending:** Further improvements of continuous performance trending, anomaly detection and analysis.

3. TEST FRAMEWORK OPTIMIZATIONS

- **General Code Housekeeping:** Ongoing RF keywords optimizations, removal of redundant RF keywords.

2.2.2 Known Issues

List of known issues in CSIT-18.07 for VPP performance tests:

#	JiraID	Issue Description
1	CSIT-570 ¹⁸	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-1361 ¹⁹	High failure rate of api call sw_interface_set_flags [admin-up link-up]. Failure rate: 30-40% of tests failing due to interfaces not in link-up state after API call sw_interface_set_flags.
3	CSIT-1234 ²⁰	VPP IPsecHW scale interface mode 1core, low NDR and PDR 64B throughput in 3n-hsw testbeds, in CSIT-18.07 vs. CSIT-18.04. ip4ipsecscale1000tnl-ip4base-int 1core CSIT-18.07/18.04 relative change: NDR -31%, PDR -32%, MRR -38%.
4	CSIT-1242 ²¹	VPP xl710 ip4base test 1core, low NDR and PDR 64B throughput in 3n-hsw testbeds, in CSIT-18.07 vs. CSIT-18.04. xl710 ip4base 1core CSIT-18.07/18.04 relative change: NDR -29%, high stdev.
5	CSIT-1243 ²²	VPP nat44 base test 2core, low NDR and PDR 64B throughput in 3n-skx testbeds, compared to 3n-hsw testbeds. ip4base-nat44 2core 3n-skx/3n-hsw relative change: NDR -19%, PDR -22%.
6	CSIT-1244 ²³	VPP lispip4 base test 2core, low NDR and PDR 64B throughput in 3n-skx testbeds, compared to 3n-hsw testbeds. ip4lispip4-ip4base 2core 3n-skx/3n-hsw relative change: NDR -11%, PDR -18%.
7	CSIT-1245 ²⁴	VPP srv6proxy-stat and srv6proxy-masq, much higher NDR and PDR 78B throughput in 3n-hsw testbeds, in CSIT-18.07 vs. CSIT-18.04. Due to wrong test suite configuration in dynamic-proxy mode. Artefact of suite code refactoring.

¹⁸ <https://jira.fd.io/browse/CSIT-570>

¹⁹ <https://jira.fd.io/browse/VPP-1361>

²⁰ <https://jira.fd.io/browse/CSIT-1234>

²¹ <https://jira.fd.io/browse/CSIT-1242>

²² <https://jira.fd.io/browse/CSIT-1243>

²³ <https://jira.fd.io/browse/CSIT-1244>

²⁴ <https://jira.fd.io/browse/CSIT-1245>

2.3 Packet Throughput

Throughput graphs are generated by multiple executions of the same performance tests across physical testbeds hosted LF FD.io labs: 3n-hsw, 2n-skx, 2n-skx. Box-and-Whisker plots are used to display variations in measured throughput values, without making any assumptions of the underlying statistical distribution.

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.

Note: Test results have been generated by [FD.io test executor vpp performance job 3n-hsw](#)²⁵, [FD.io test executor vpp performance job 3n-skx](#)²⁶ and [FD.io test executor vpp performance job 2n-skx](#)²⁷ with RF result files csit-vpp-perf-1807-*.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 . This is due to not all test executions completing successfully due to a known [VPP-1361](#)²⁸ issue sporadically impacting physical interface up/down state.

²⁵ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

²⁶ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-skx>

²⁷ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-2n-skx>

²⁸ <https://jira.fd.io/browse/VPP-1361>

2.3.1 L2 Ethernet Switching

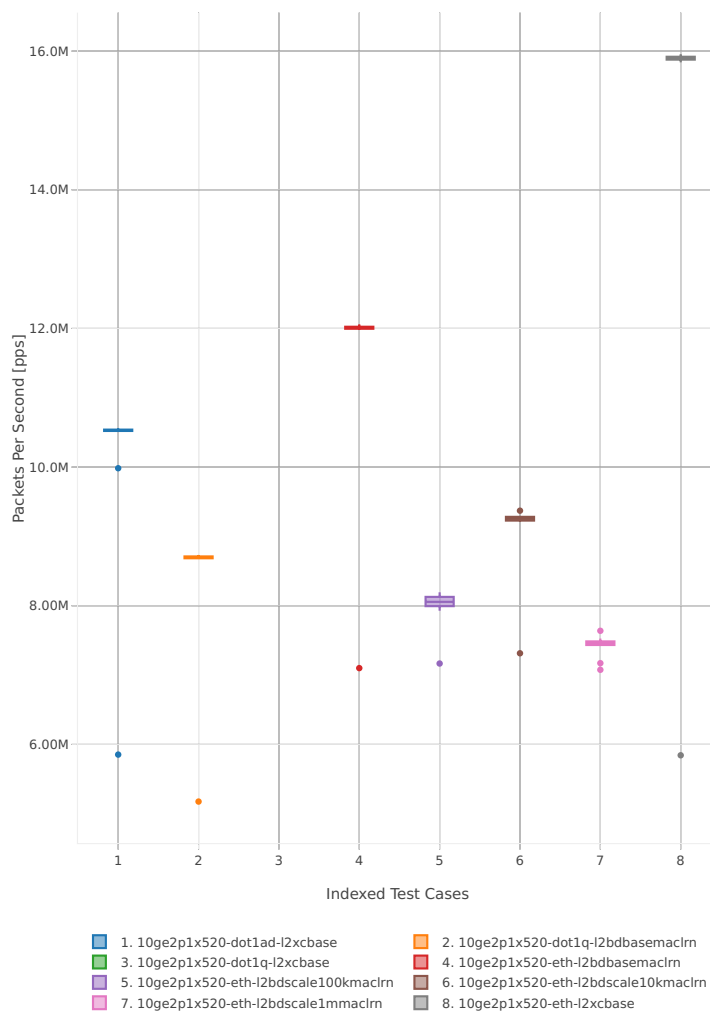
Following sections include summary graphs of VPP Phy-to-Phy performance with L2 Ethernet switching, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s),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

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)²⁹.

3n-hsw-x520

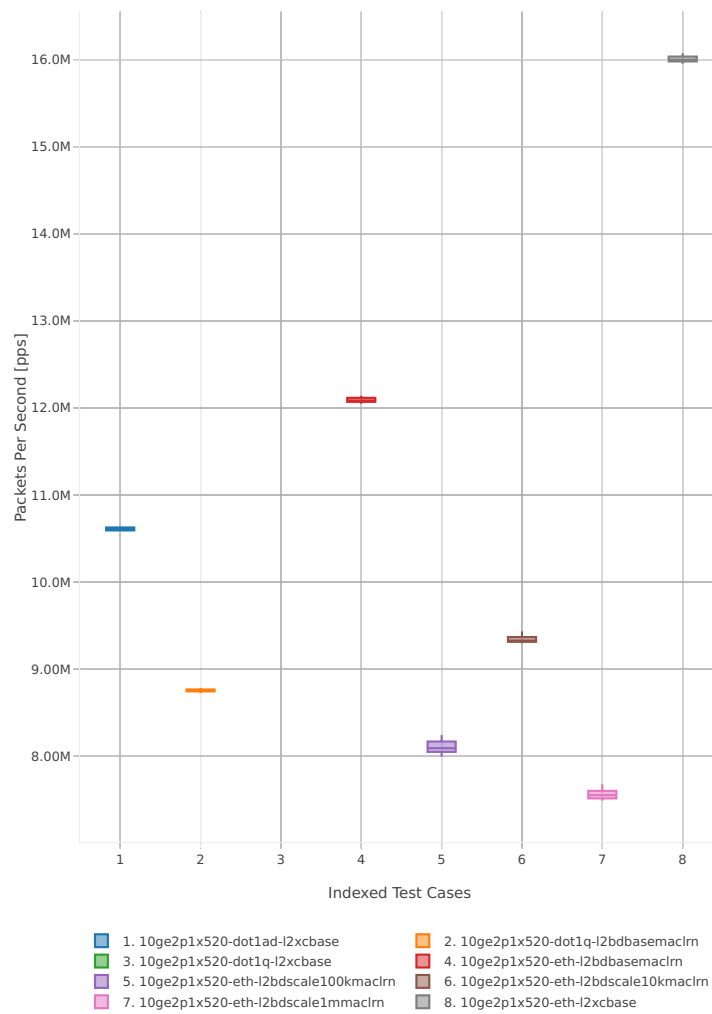
64b-1t1c-base_and_scale

Throughput: l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



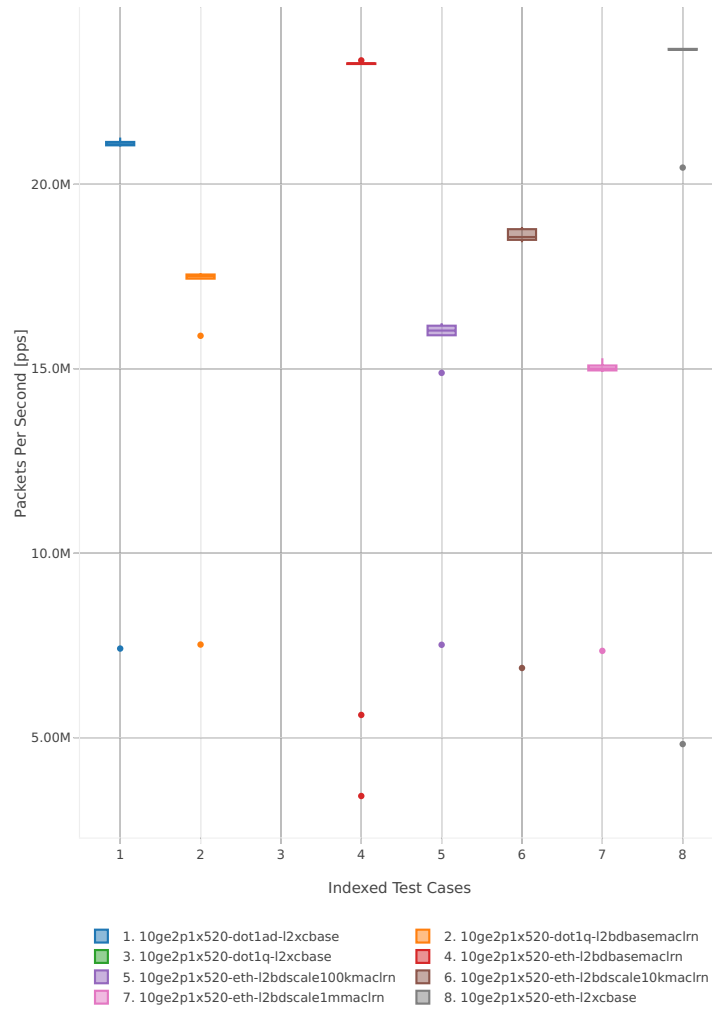
²⁹ <https://git.fd.io/csit/tree/tests/vpp/perf/l2?h=rls1807>

Throughput: l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

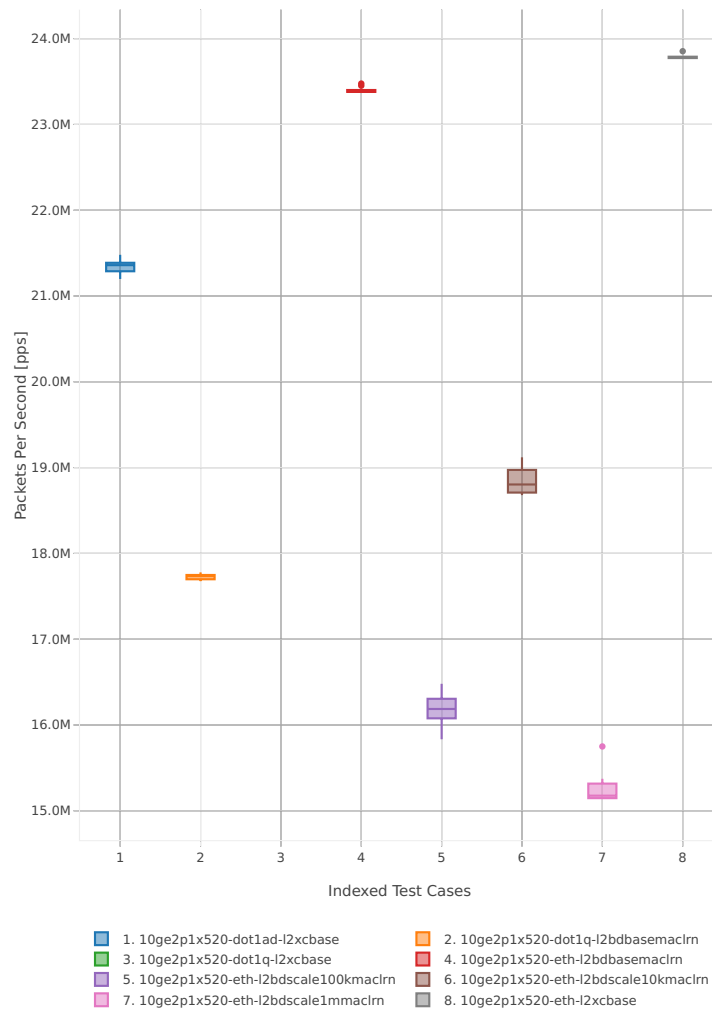


64b-2t2c-base_and_scale

Throughput: l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-ndr

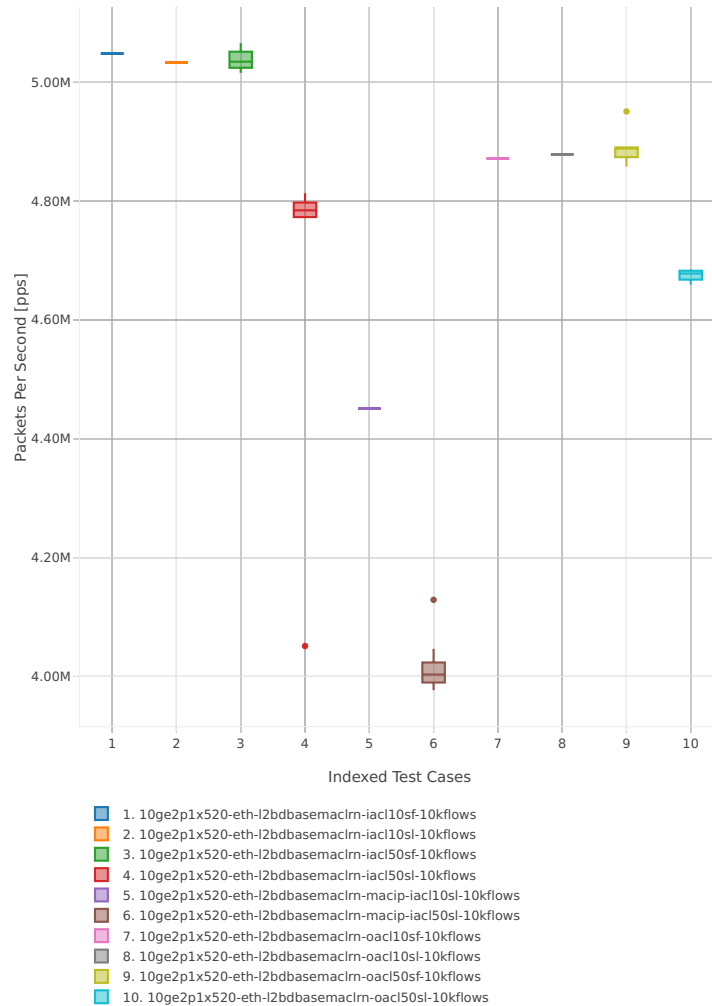


Throughput: l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-pdr

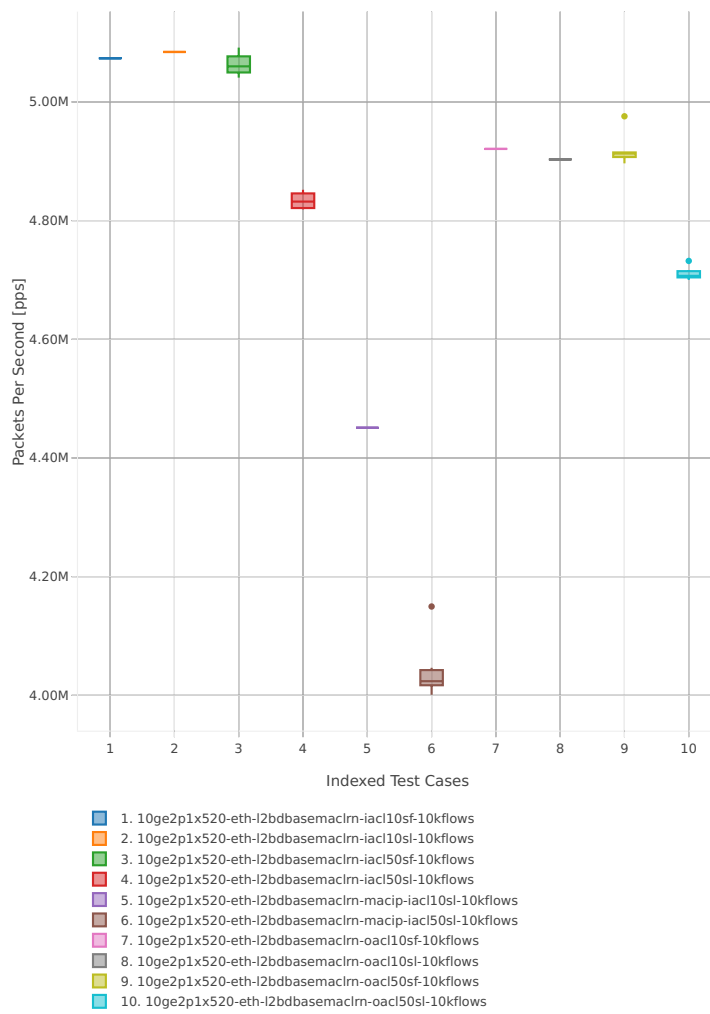


64b-1t1c-features

Throughput: l2sw-3n-hsw-x520-64b-1t1c-features-ndr

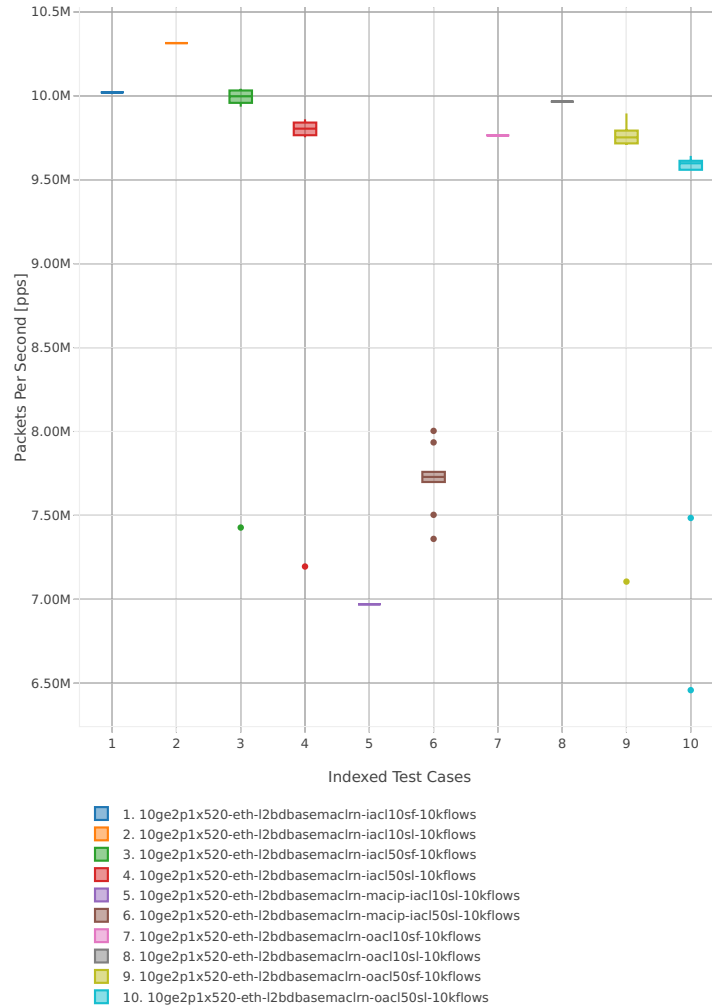


Throughput: l2sw-3n-hsw-x520-64b-1t1c-features-pdr

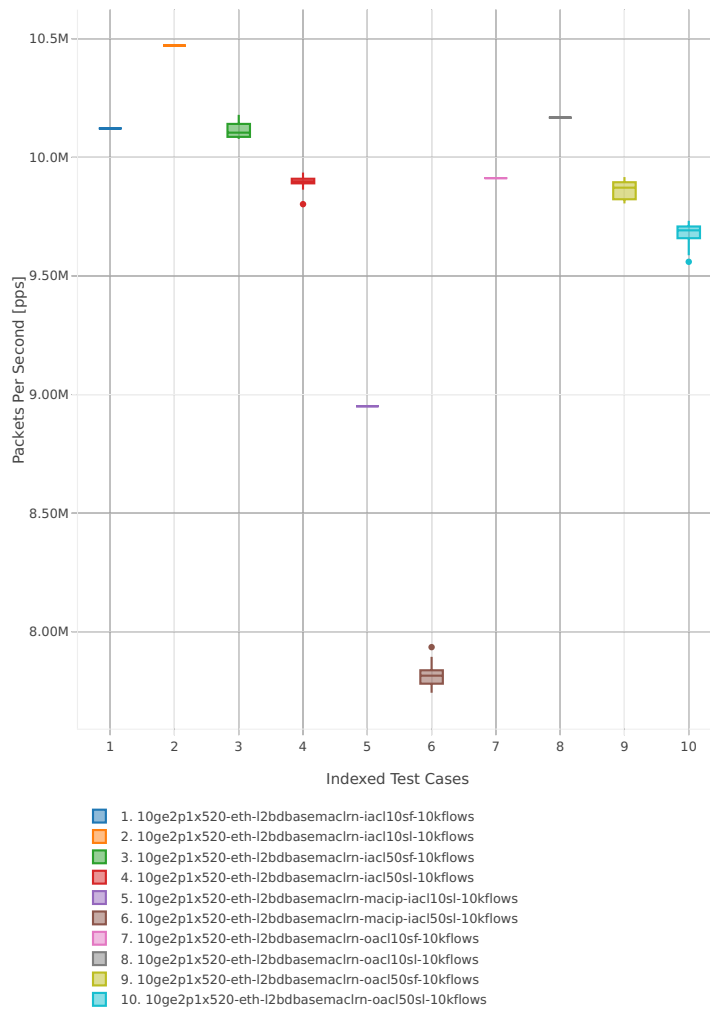


64b-2t2c-features

Throughput: l2sw-3n-hsw-x520-64b-2t2c-features-ndr



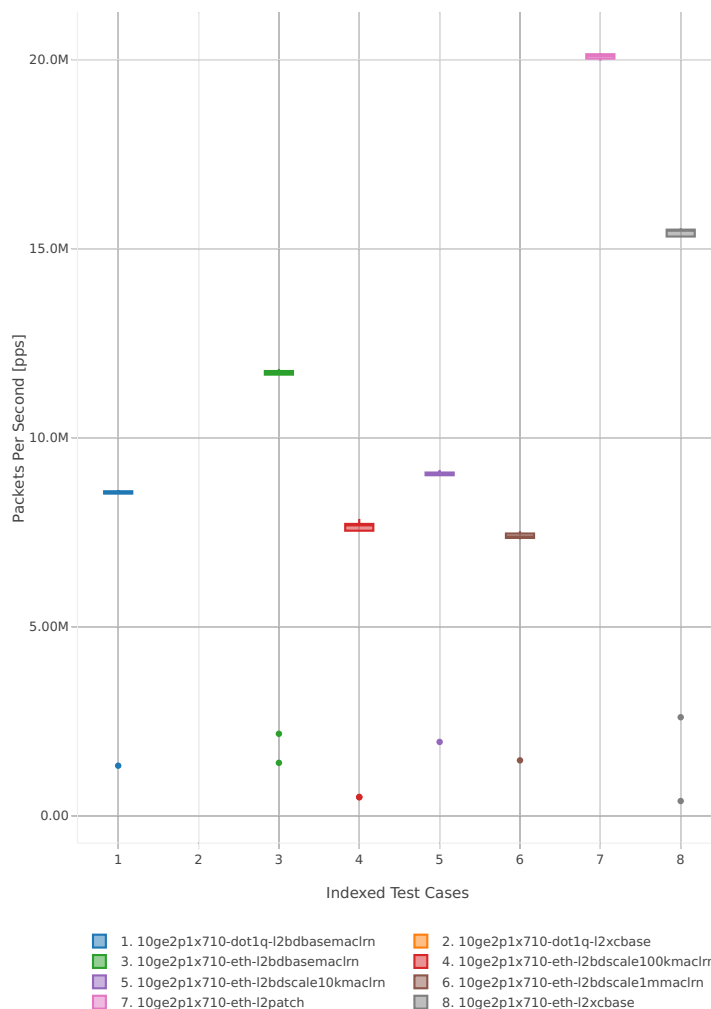
Throughput: l2sw-3n-hsw-x520-64b-2t2c-features-pdr



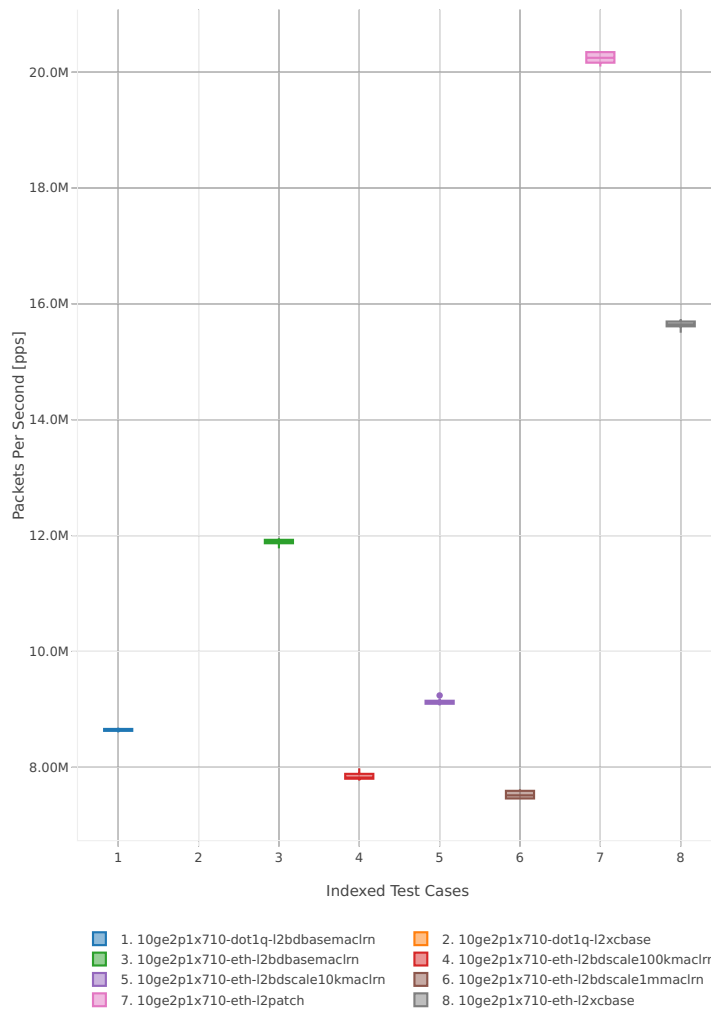
3n-hsw-x710

64b-1t1c-base_and_scale

Throughput: l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

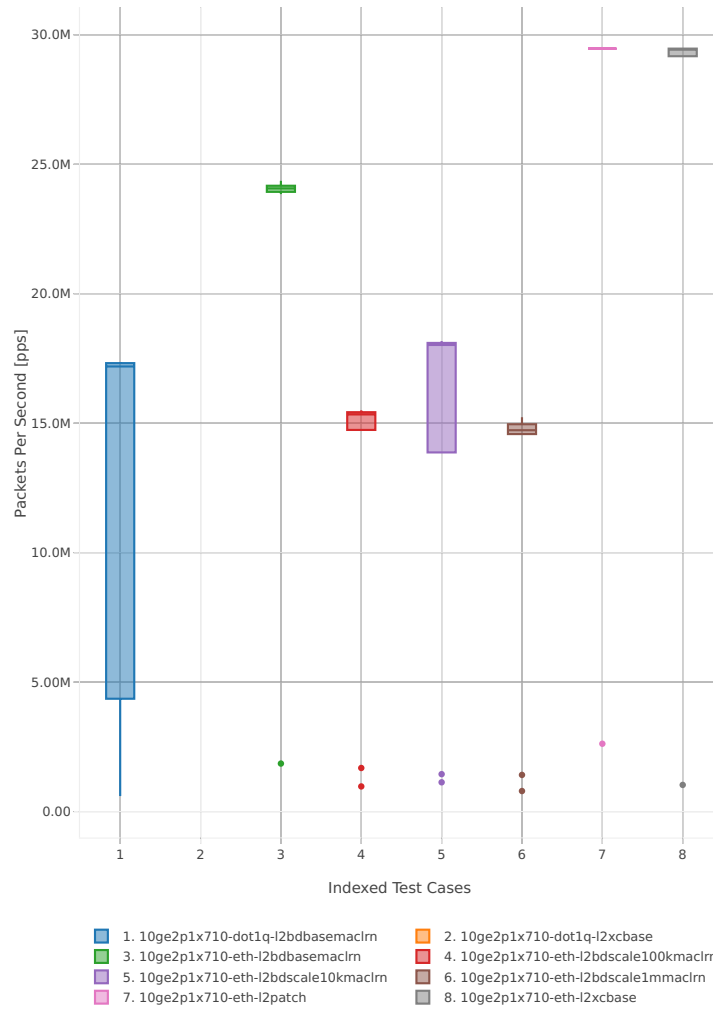


Throughput: l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

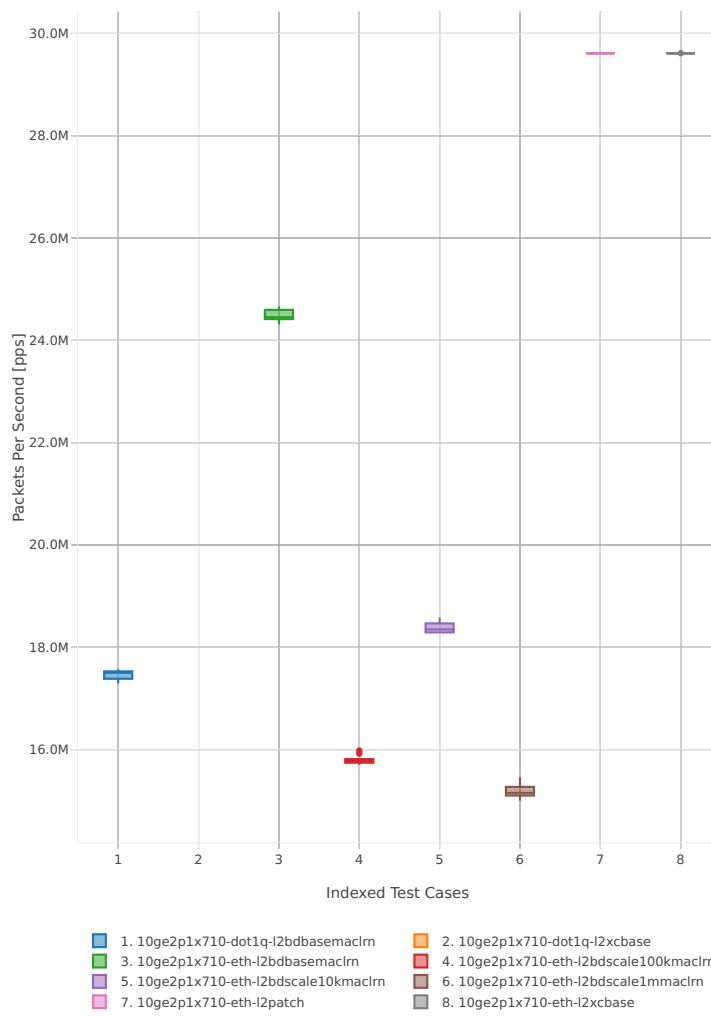


64b-2t2c-base_and_scale

Throughput: l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-ndr

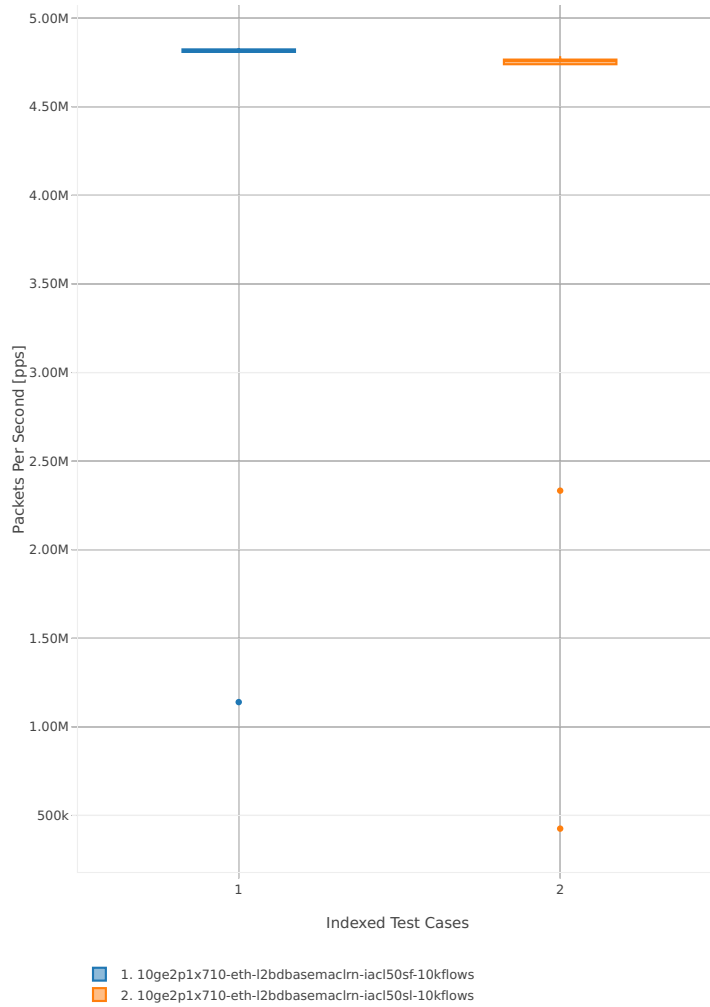


Throughput: l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-pdr

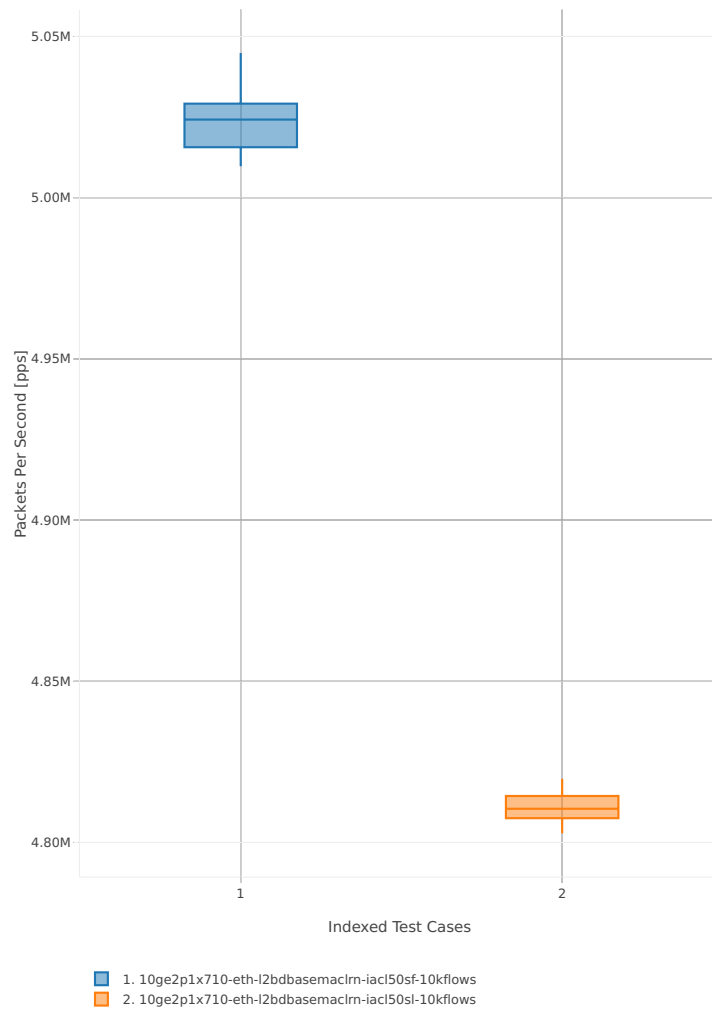


64b-1t1c-features

Throughput: l2sw-3n-hsw-x710-64b-1t1c-features-ndr

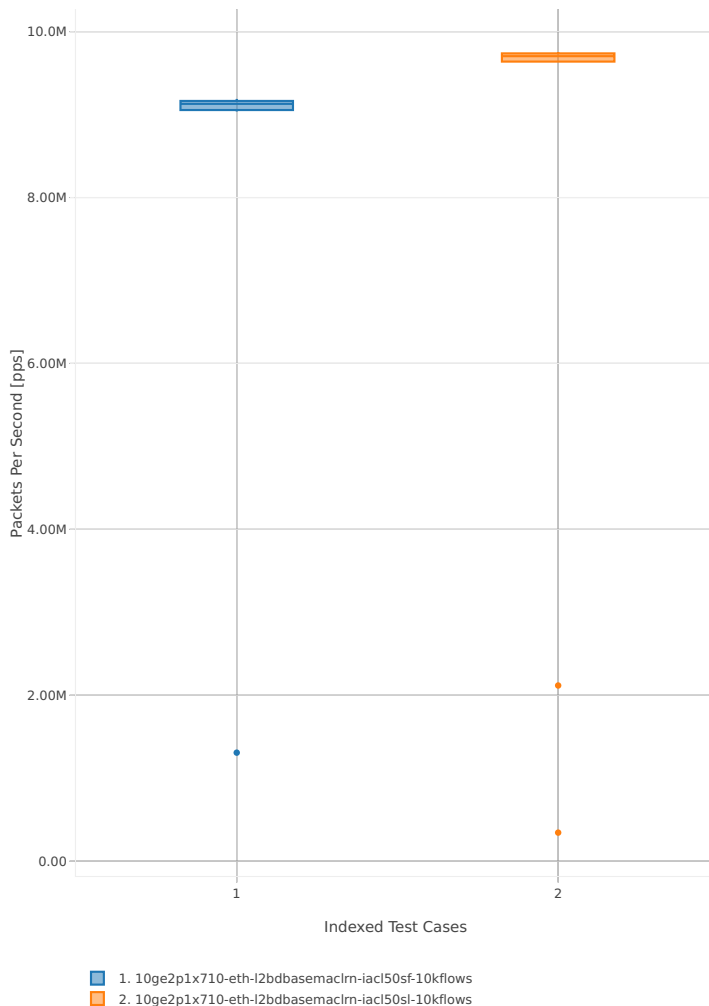


Throughput: l2sw-3n-hsw-x710-64b-1t1c-features-pdr

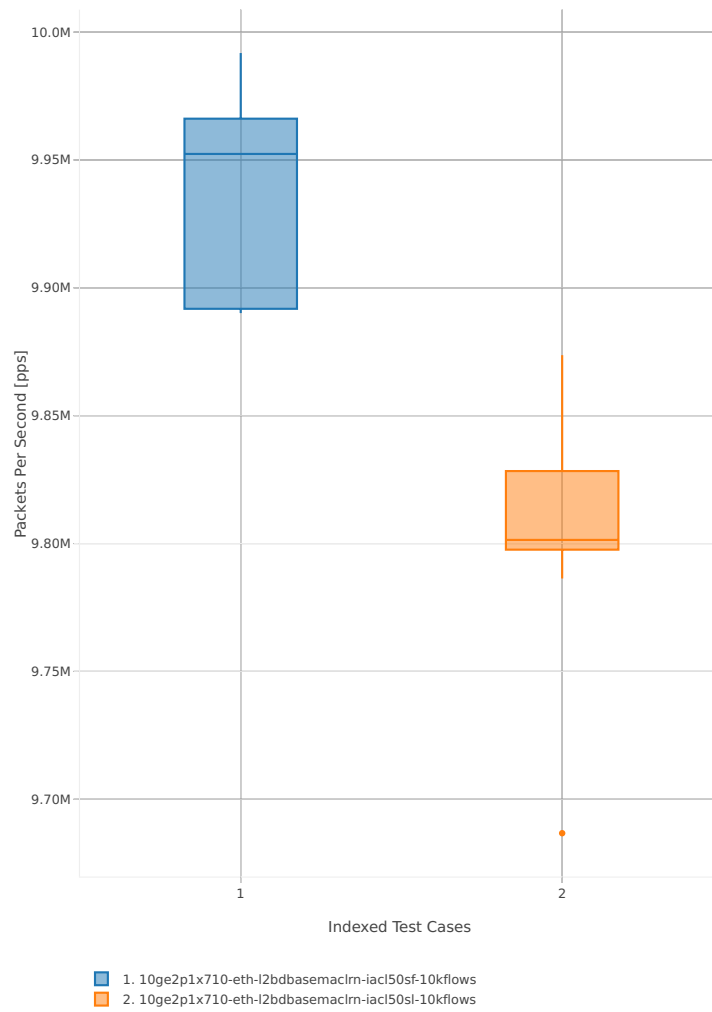


64b-2t2c-features

Throughput: l2sw-3n-hsw-x710-64b-2t2c-features-ndr



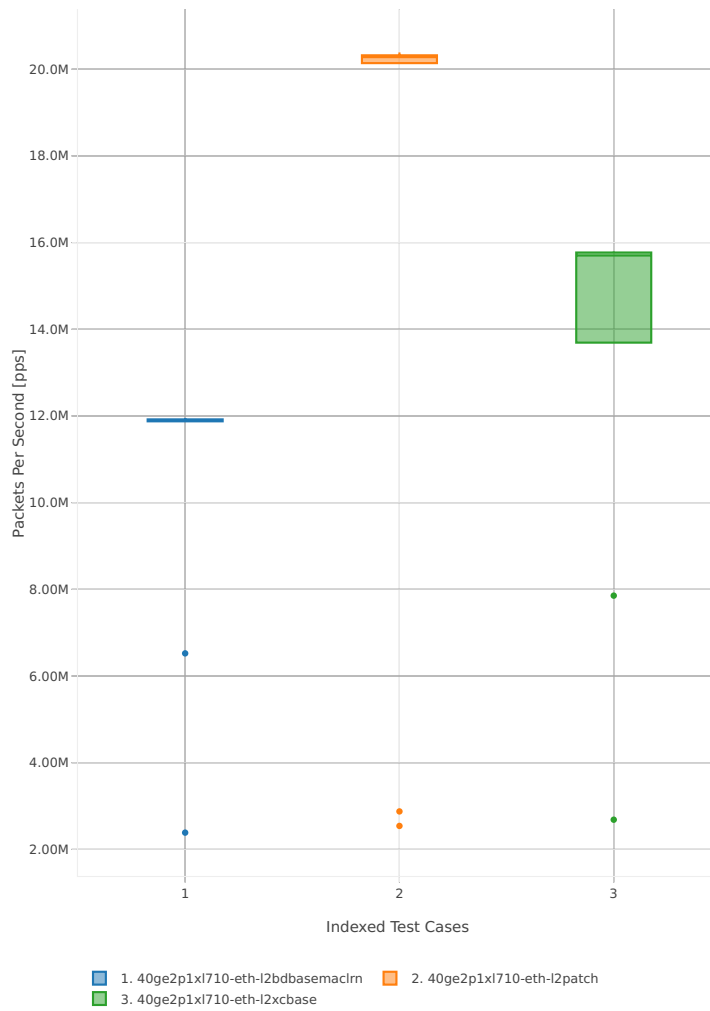
Throughput: l2sw-3n-hsw-x710-64b-2t2c-features-pdr



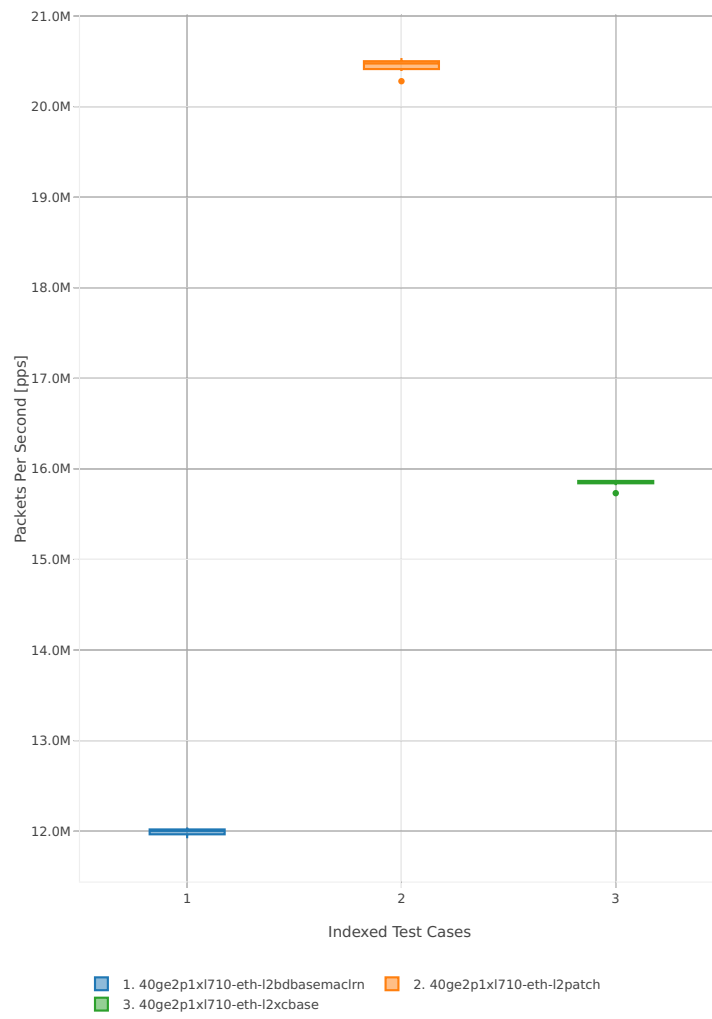
3n-hsw-xl710

64b-1t1c-base_and_scale

Throughput: l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr

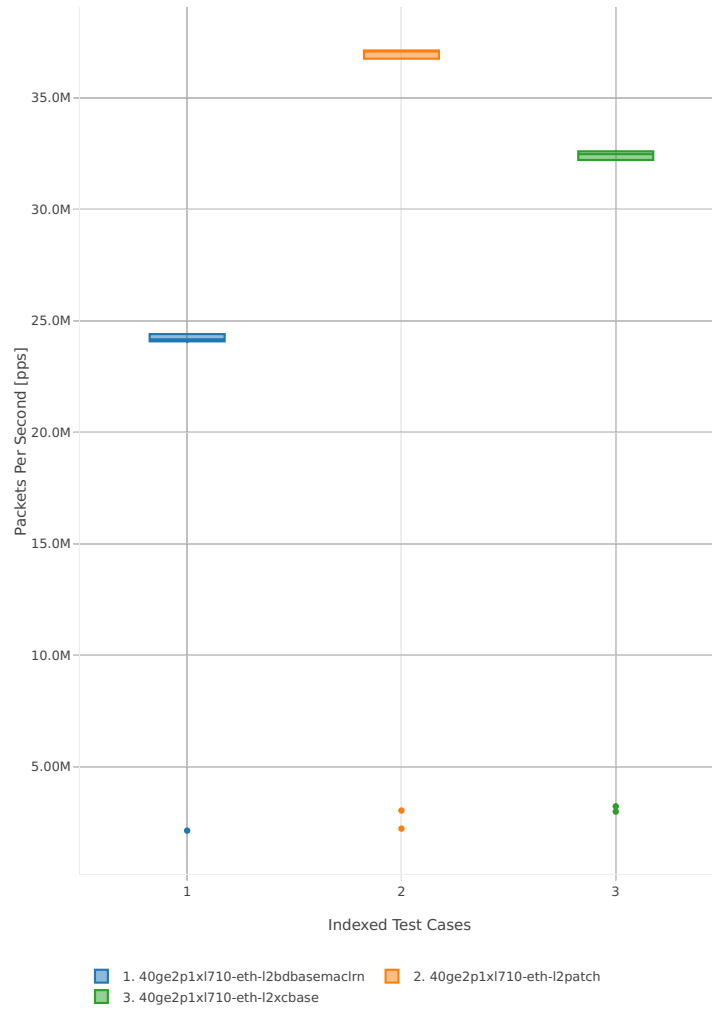


Throughput: l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

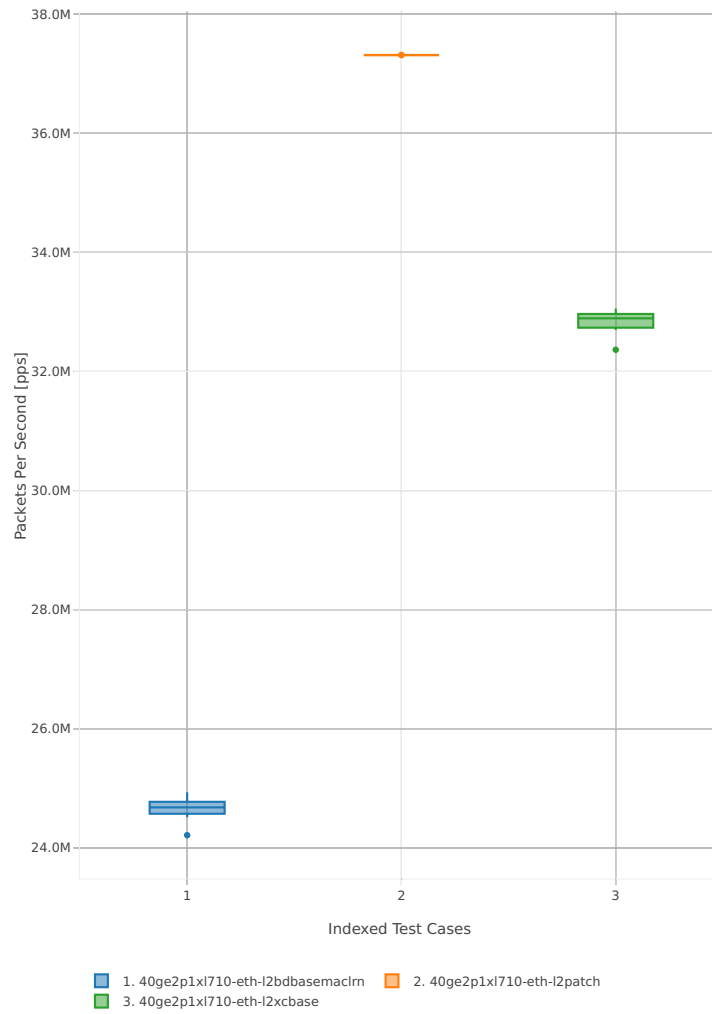


64b-2t2c-base_and_scale

Throughput: l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



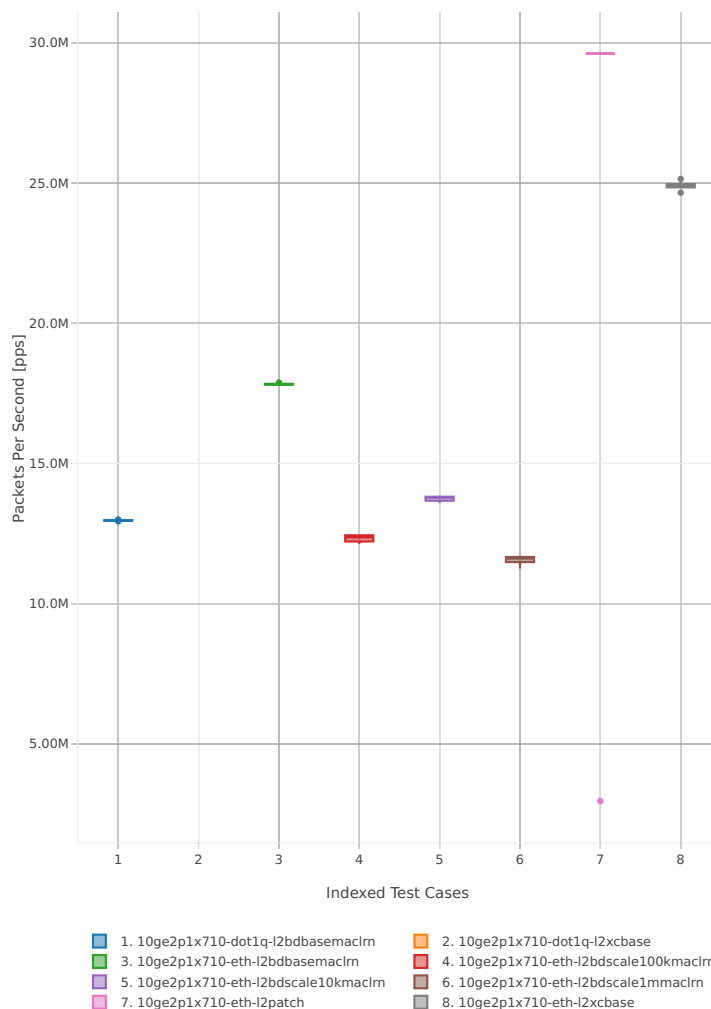
Throughput: l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr



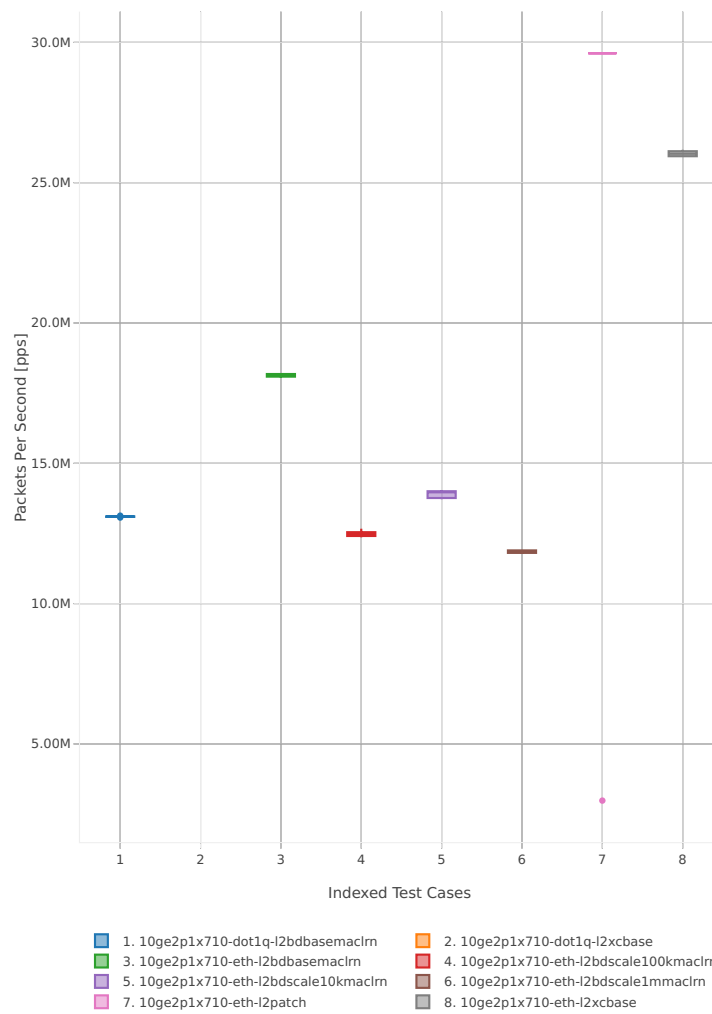
3n-skx-x710

64b-2t1c-base_and_scale

Throughput: l2sw-3n-skx-x710-64b-2t1c-base_and_scale-ndr

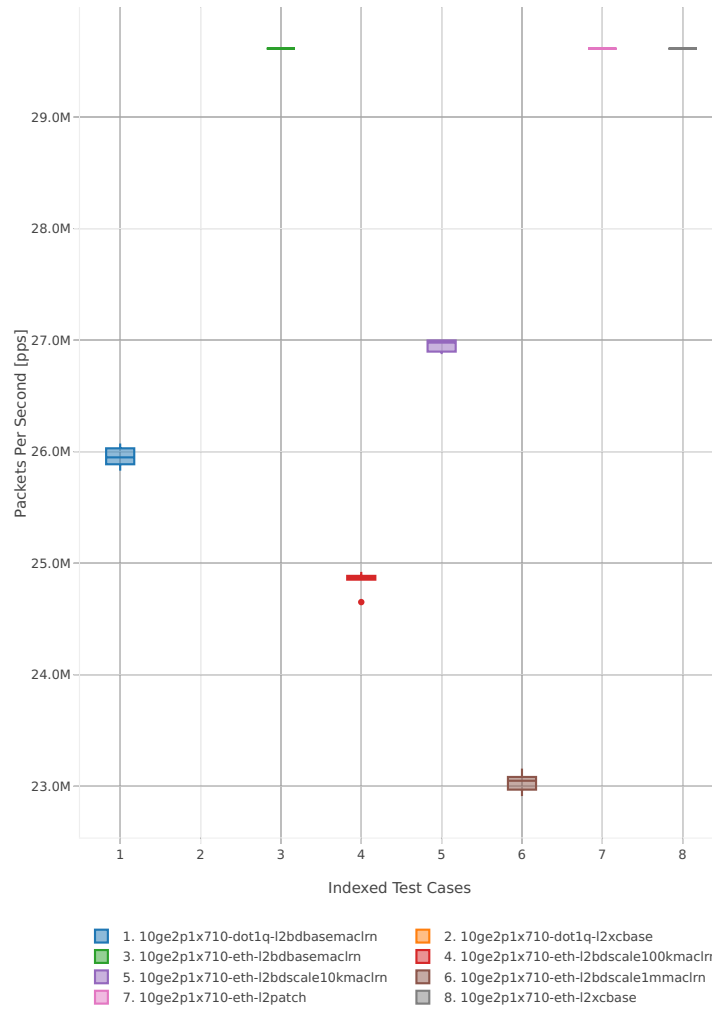


Throughput: l2sw-3n-skx-x710-64b-2t1c-base_and_scale-pdr

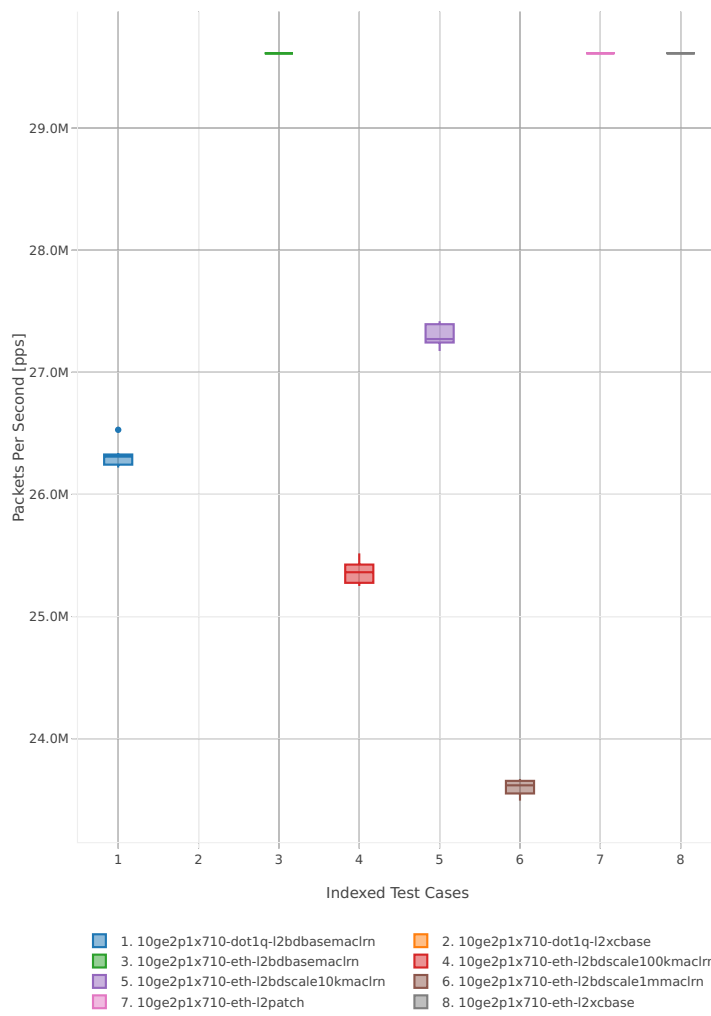


64b-4t2c-base_and_scale

Throughput: l2sw-3n-skx-x710-64b-4t2c-base_and_scale-ndr

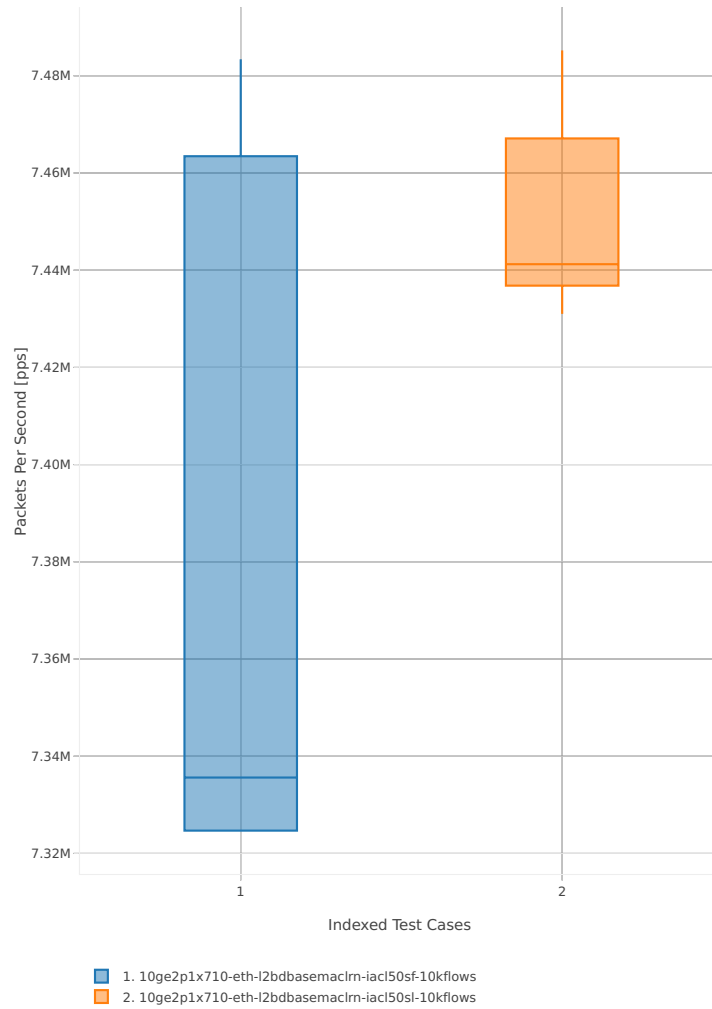


Throughput: l2sw-3n-skx-x710-64b-4t2c-base_and_scale-pdr

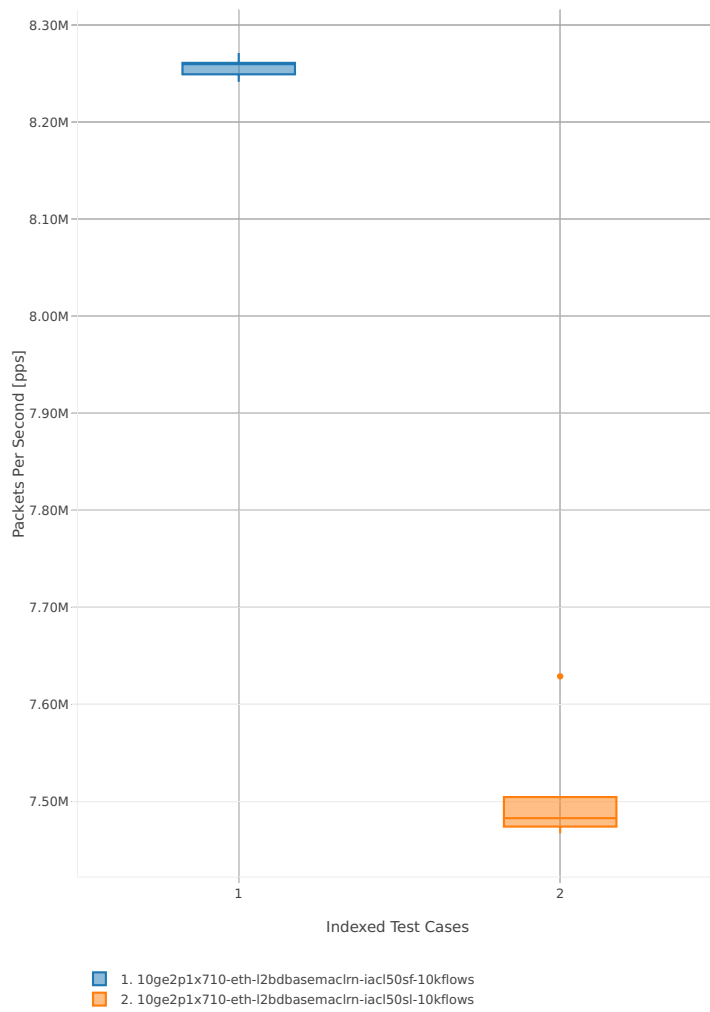


64b-2t1c-features

Throughput: l2sw-3n-skx-x710-64b-2t1c-features-ndr

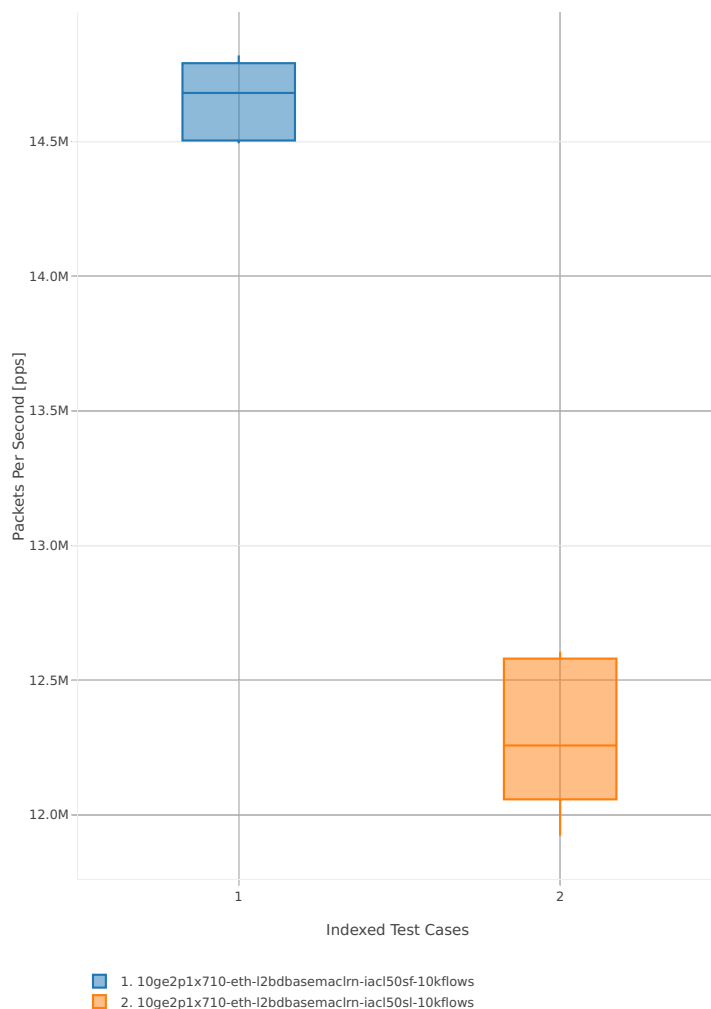


Throughput: l2sw-3n-skx-x710-64b-2t1c-features-pdr

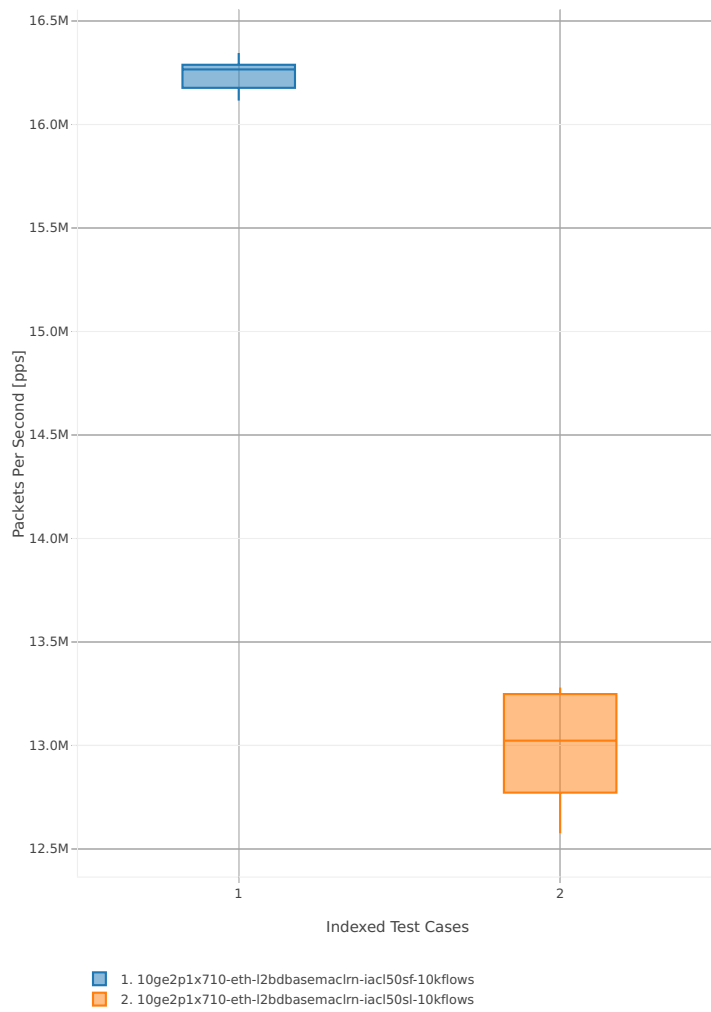


64b-4t2c-features

Throughput: l2sw-3n-skx-x710-64b-4t2c-features-ndr



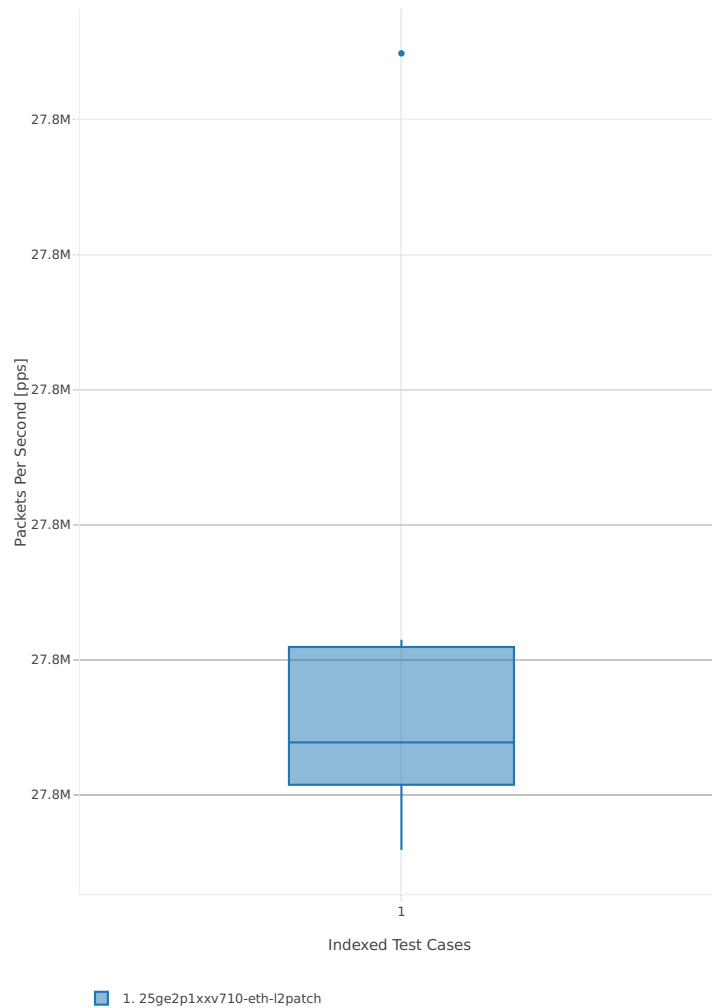
Throughput: l2sw-3n-skx-x710-64b-4t2c-features-pdr



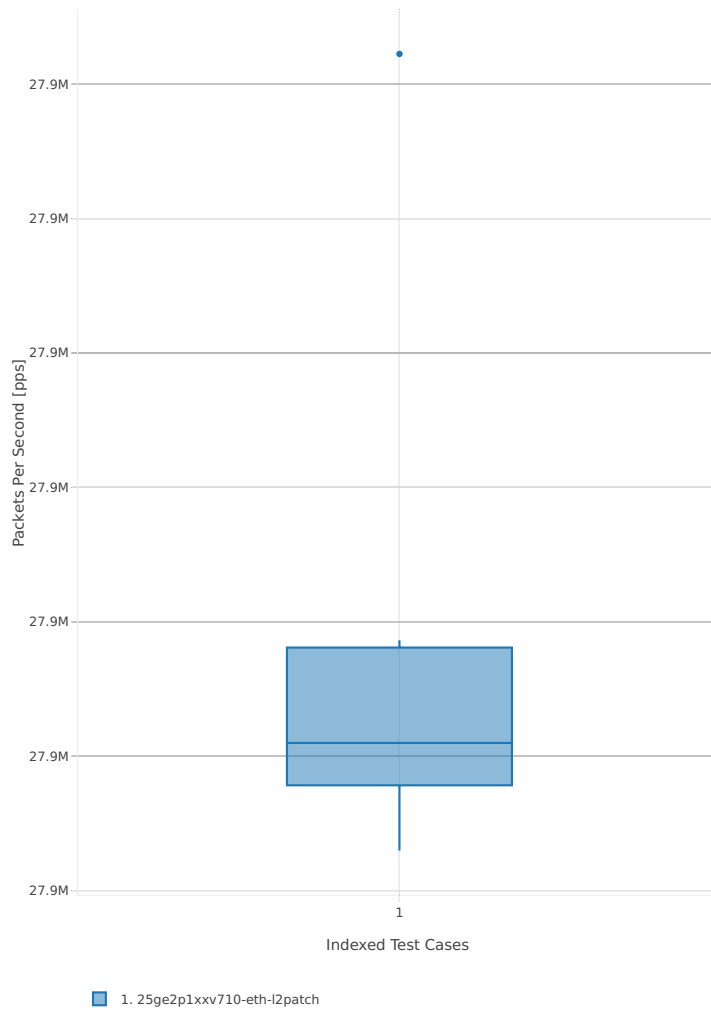
3n-skx-xxv710

64b-2t1c-base_and_scale

Throughput: l2sw-3n-skx-xxv710-64b-2t1c-base_and_scale-ndr

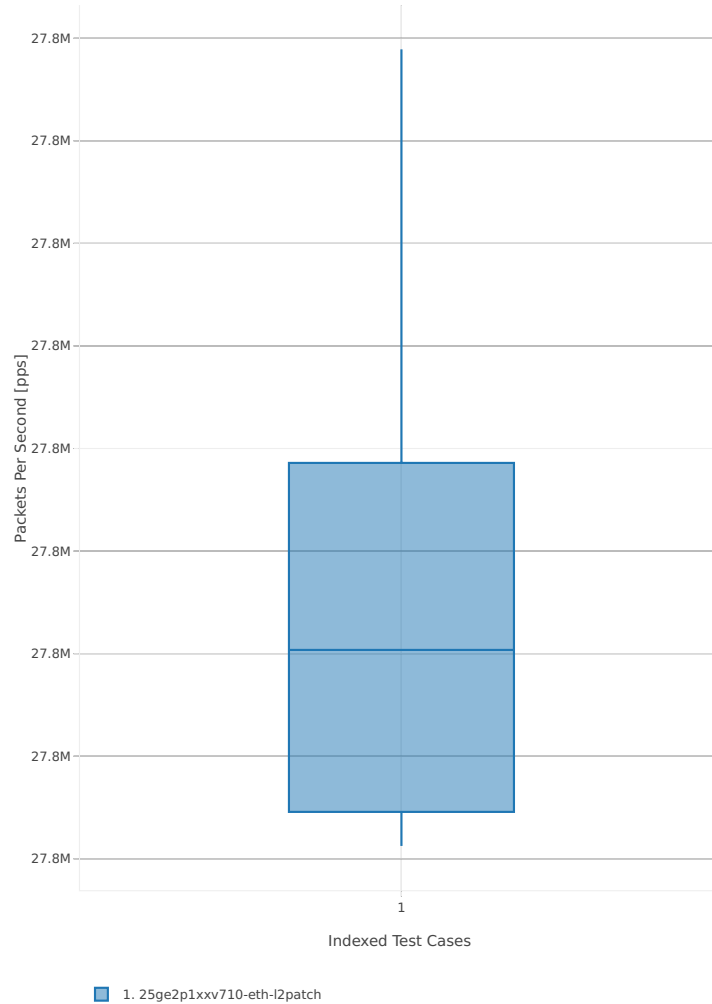


Throughput: l2sw-3n-skx-xxv710-64b-2t1c-base_and_scale-pdr

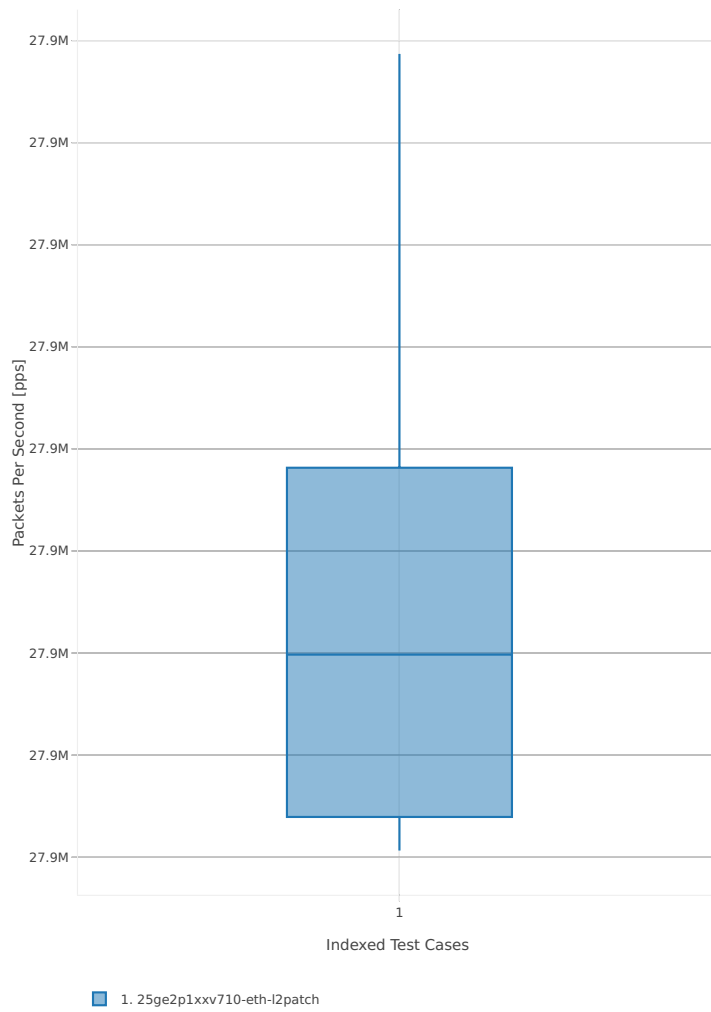


64b-4t2c-base_and_scale

Throughput: l2sw-3n-skx-xxv710-64b-4t2c-base_and_scale-ndr



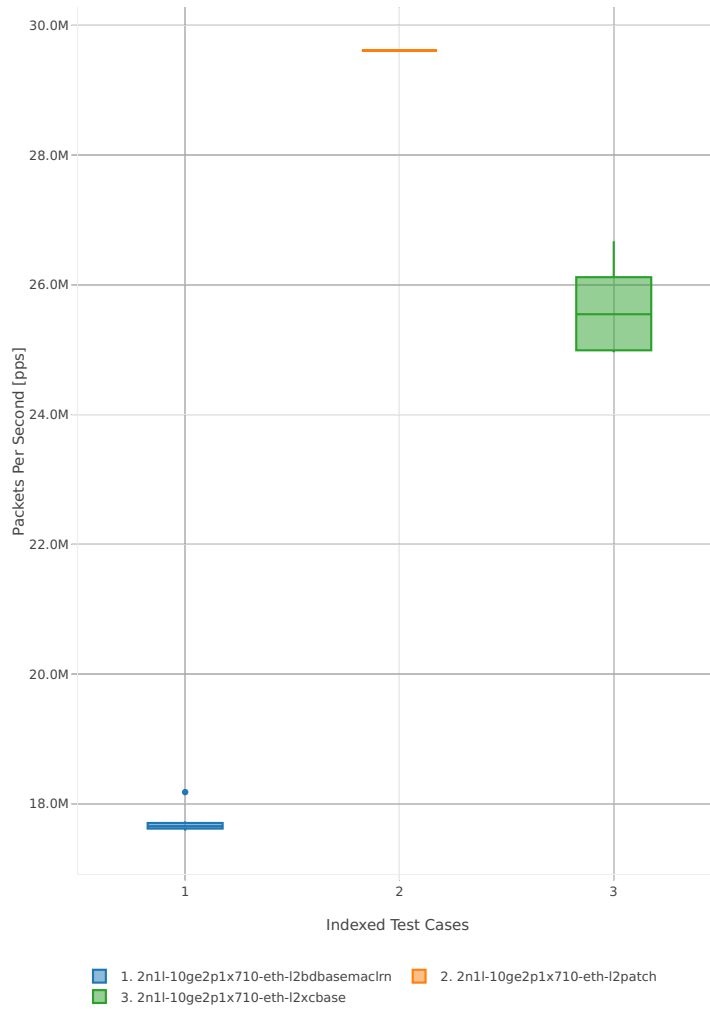
Throughput: l2sw-3n-skx-xxv710-64b-4t2c-base_and_scale-pdr



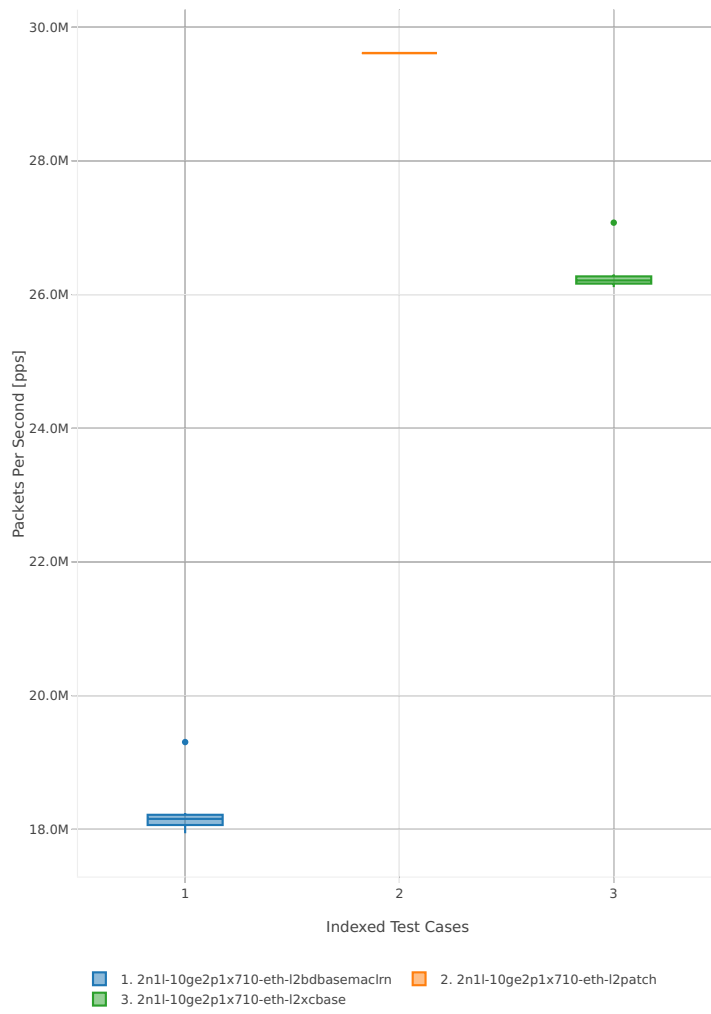
2n-skx-x710

64b-2t1c-base_and_scale

Throughput: l2sw-2n-skx-x710-64b-2t1c-base_and_scale-ndr

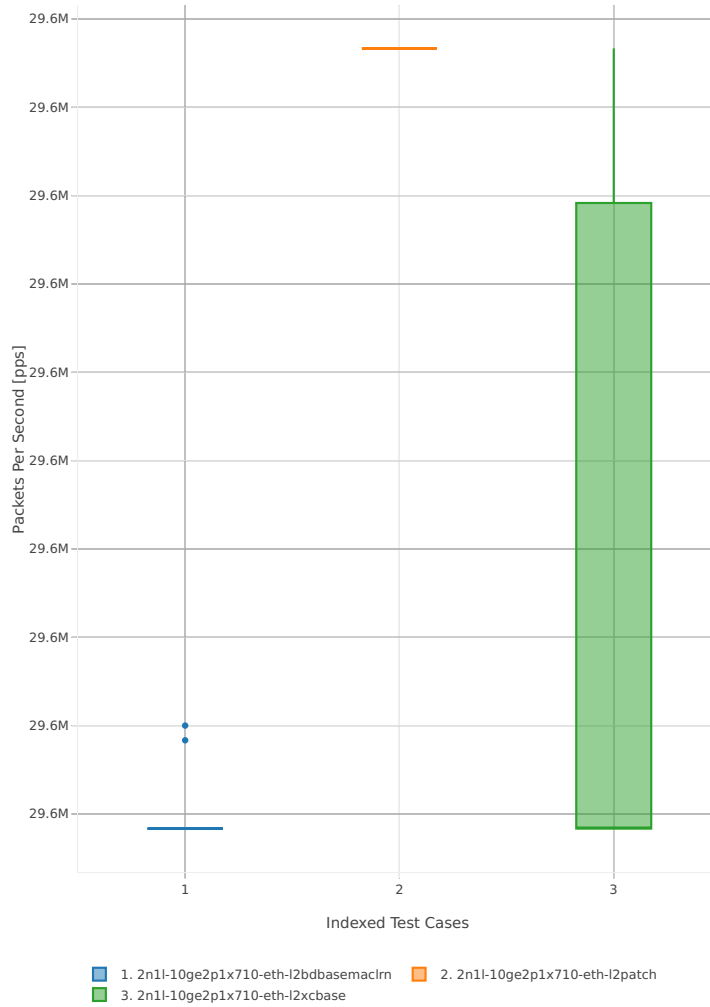


Throughput: l2sw-2n-skx-x710-64b-2t1c-base_and_scale-pdr

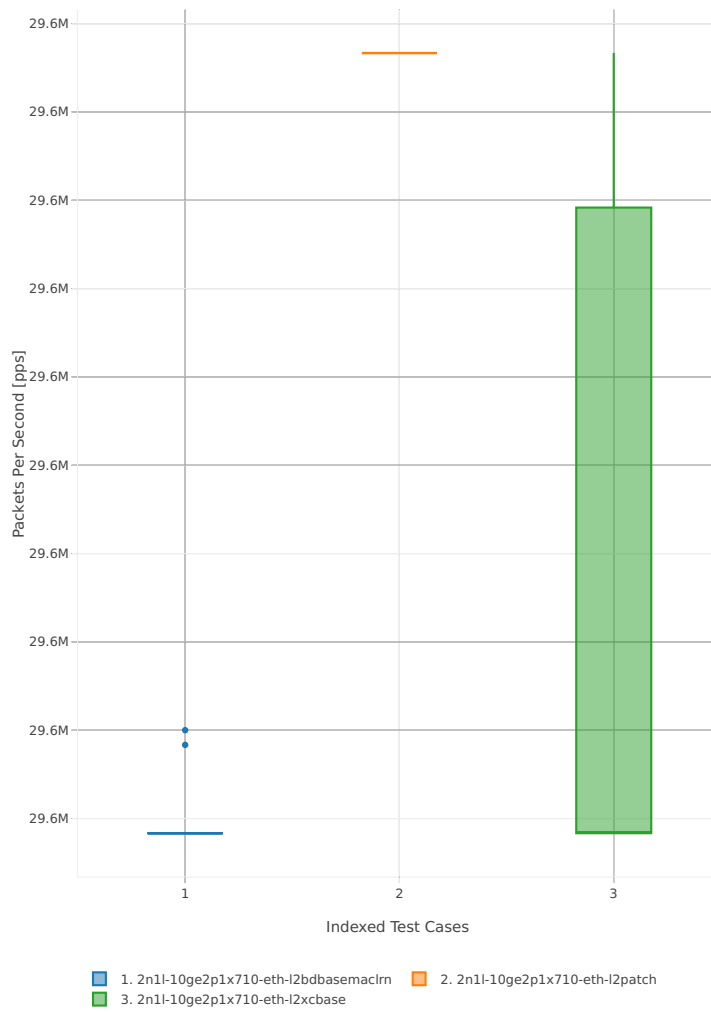


64b-4t2c-base_and_scale

Throughput: l2sw-2n-skx-x710-64b-4t2c-base_and_scale-ndr



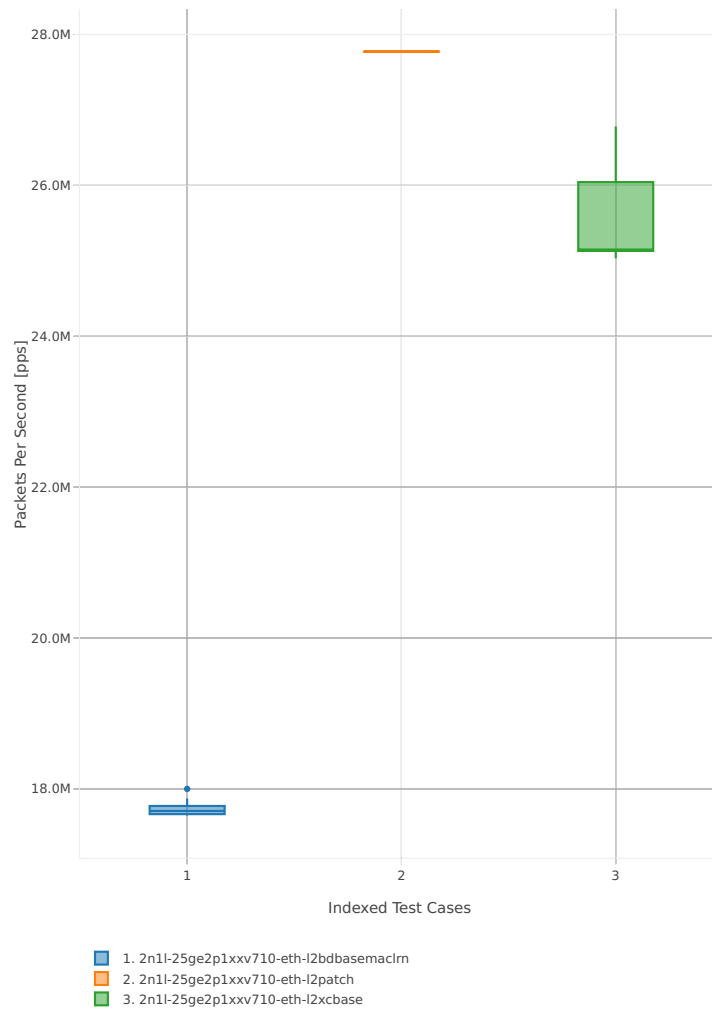
Throughput: l2sw-2n-skx-x710-64b-4t2c-base_and_scale-pdr



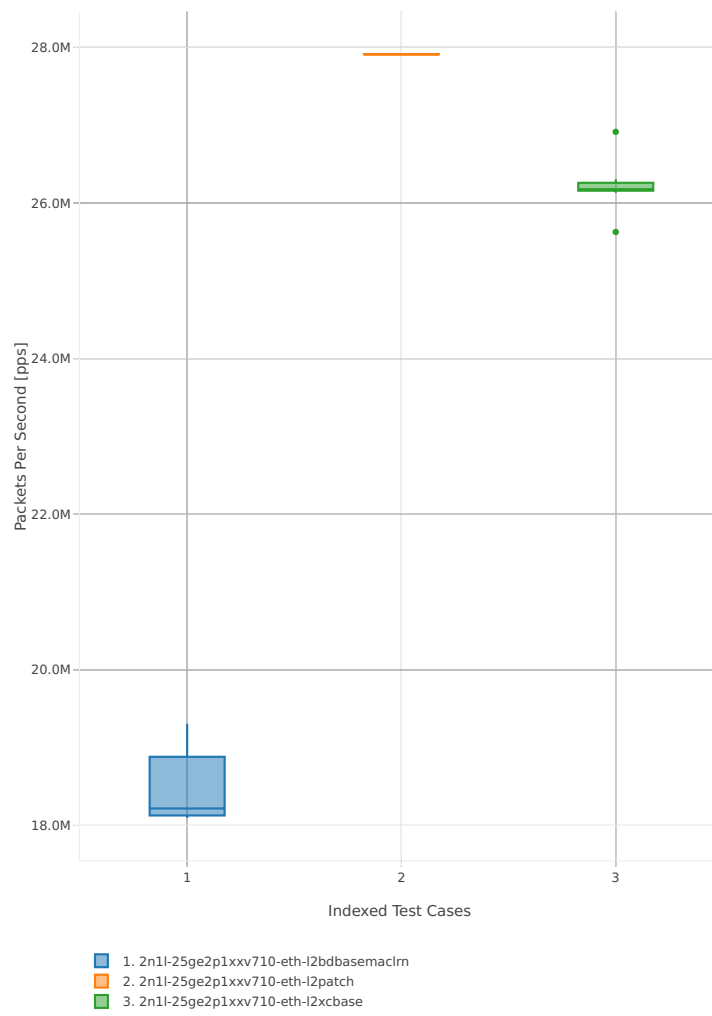
2n-skx-xxv710

64b-2t1c-base_and_scale

Throughput: l2sw-2n-skx-xxv710-64b-2t1c-base_and_scale-ndr

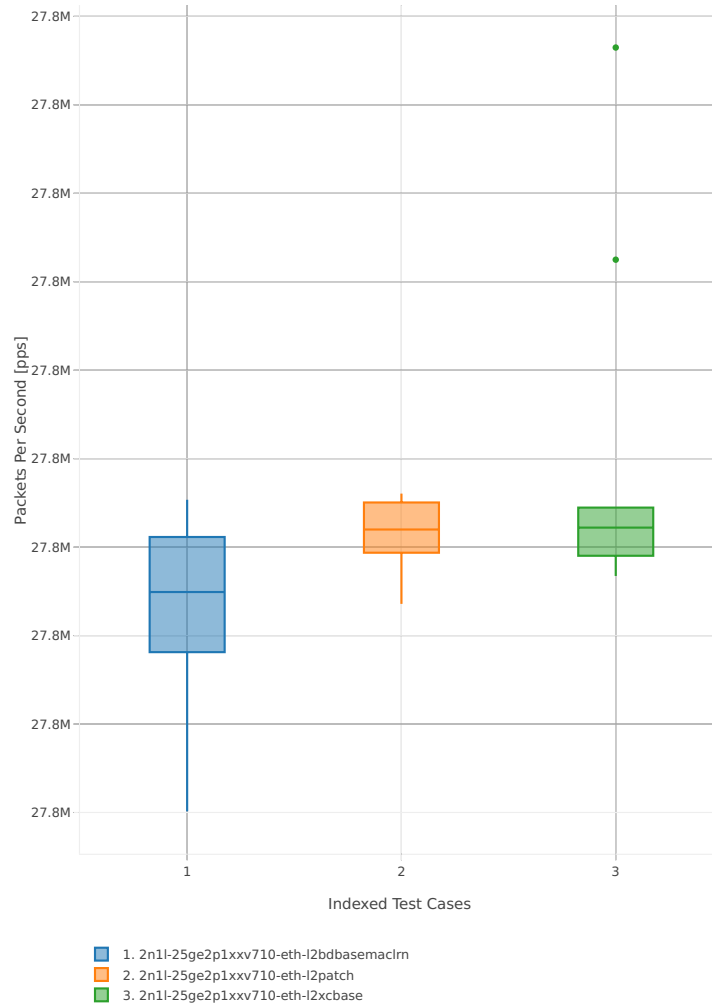


Throughput: l2sw-2n-skx-xxv710-64b-2t1c-base_and_scale-pdr

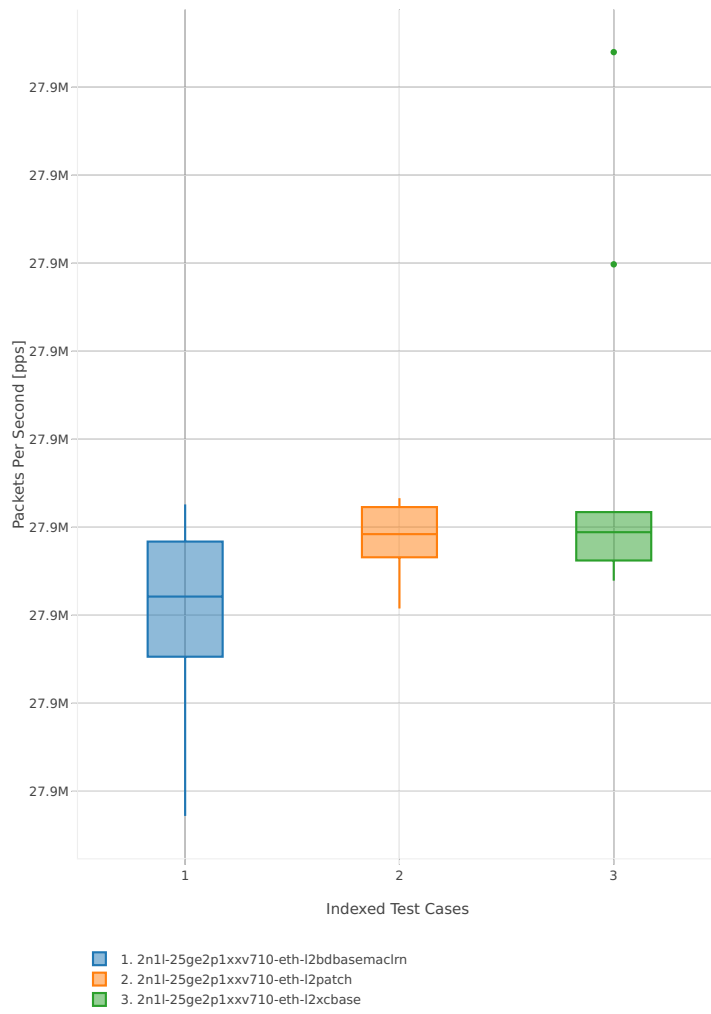


64b-4t2c-base_and_scale

Throughput: l2sw-2n-skx-xxv710-64b-4t2c-base_and_scale-ndr



Throughput: l2sw-2n-skx-xxv710-64b-4t2c-base_and_scale-pdr



2.3.2 IPv4 Routing

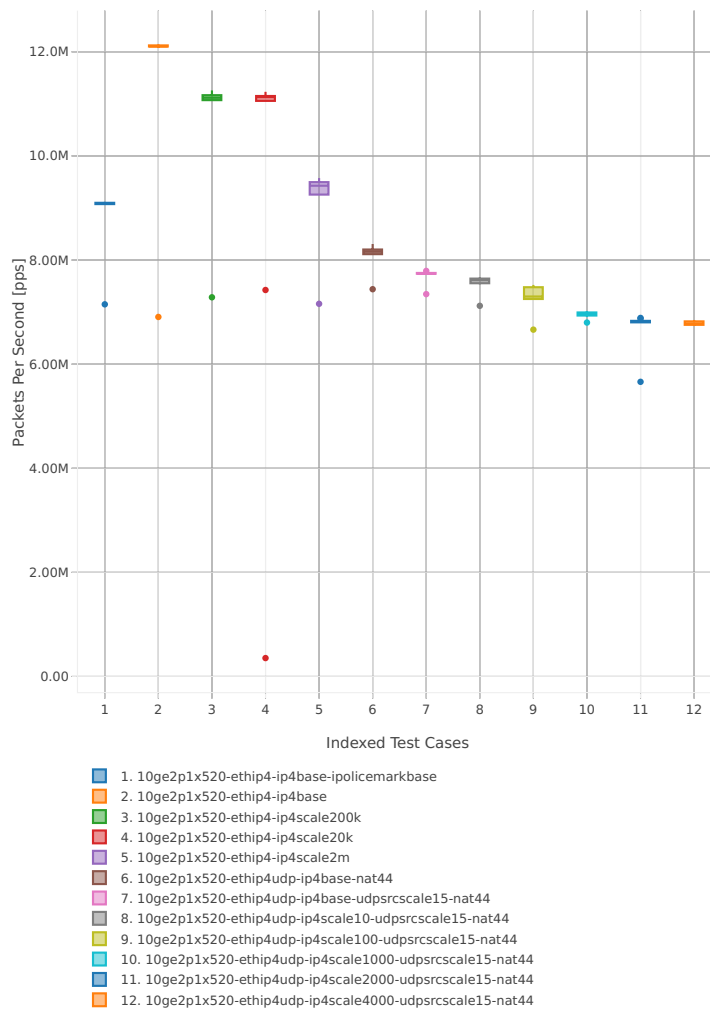
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv4 Routed-Forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s),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

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)³⁰.

3n-hsw-x520

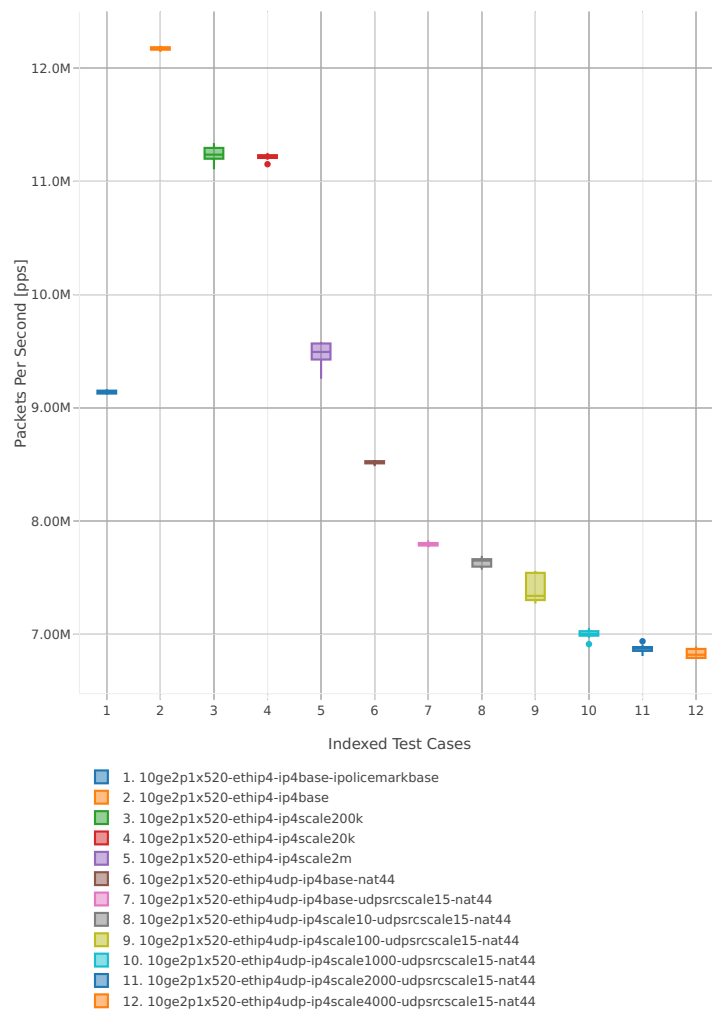
64b-1t1c-base_and_scale

Throughput: ip4-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



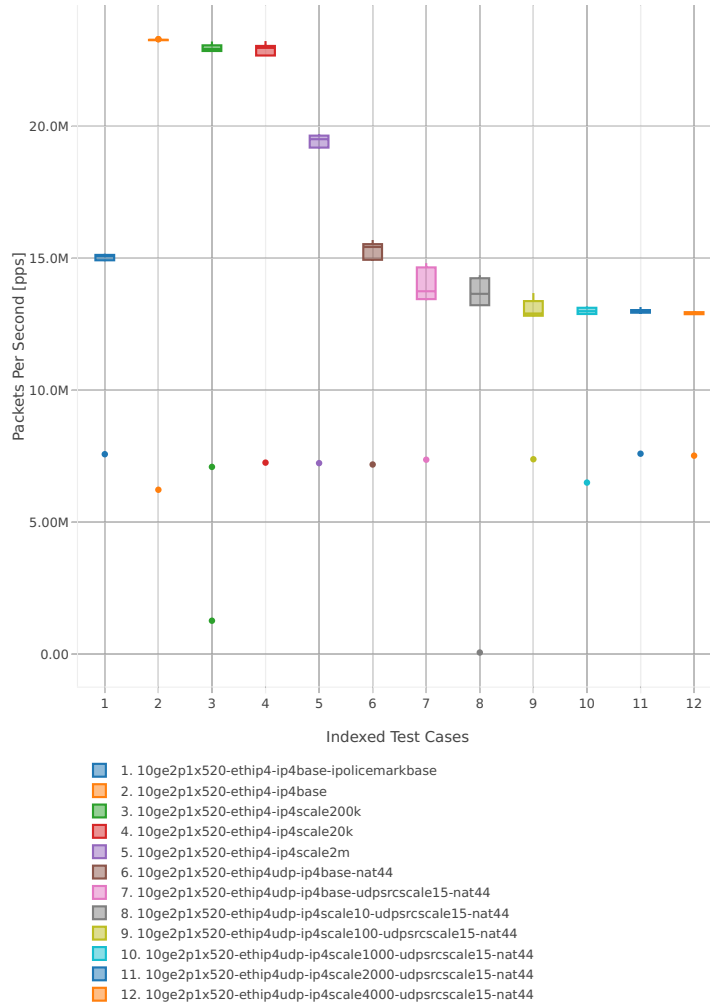
³⁰ <https://git.fd.io/csit/tree/tests/vpp/perf/ip4?h=rls1807>

Throughput: ip4-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

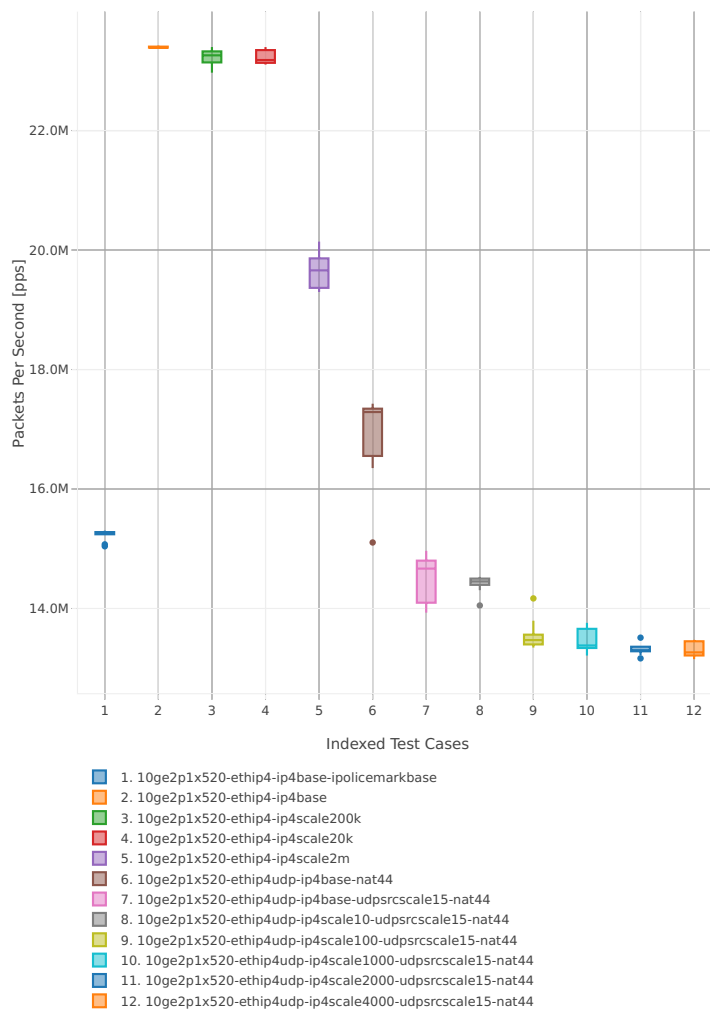


64b-2t2c-base_and_scale

Throughput: ip4-3n-hsw-x520-64b-2t2c-base_and_scale-ndr

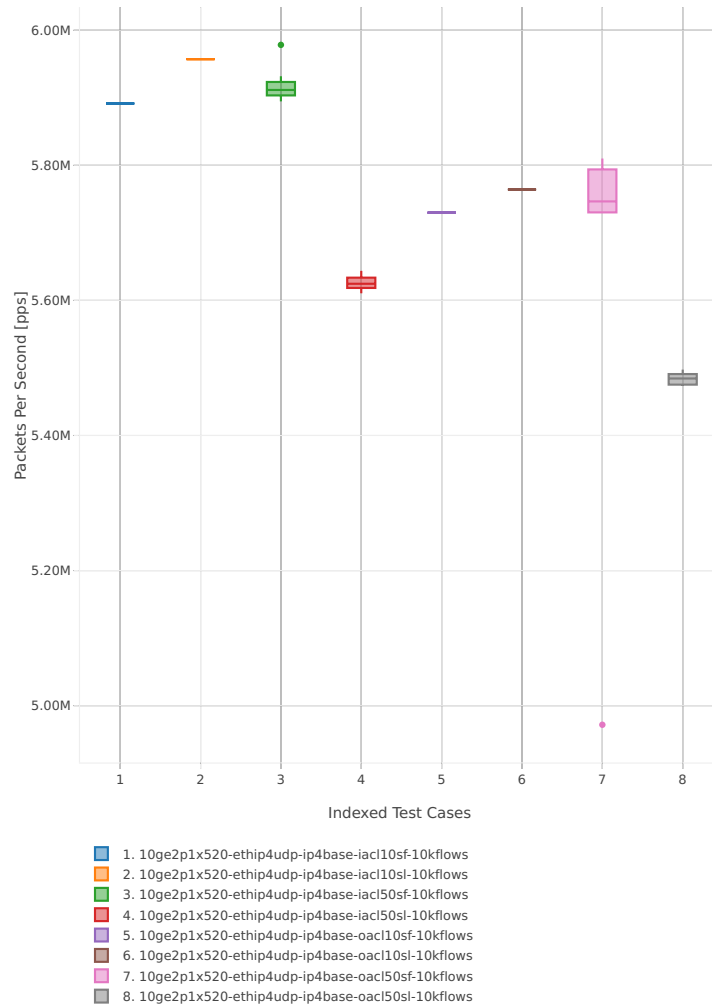


Throughput: ip4-3n-hsw-x520-64b-2t2c-base_and_scale-pdr

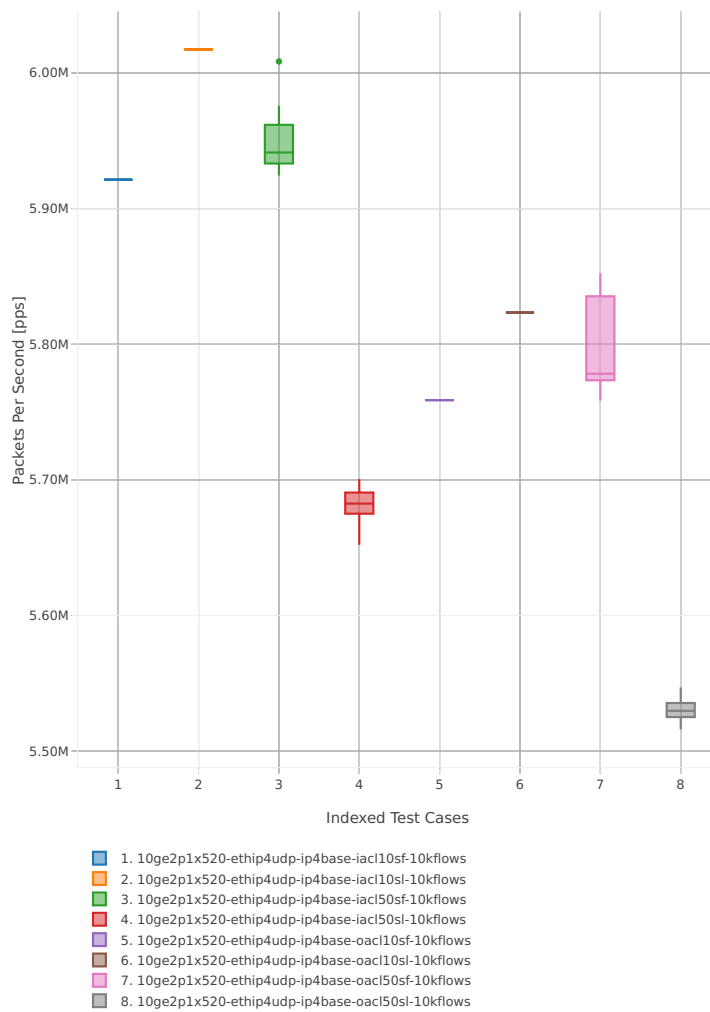


64b-1t1c-features

Throughput: ip4-3n-hsw-x520-64b-1t1c-features-ndr

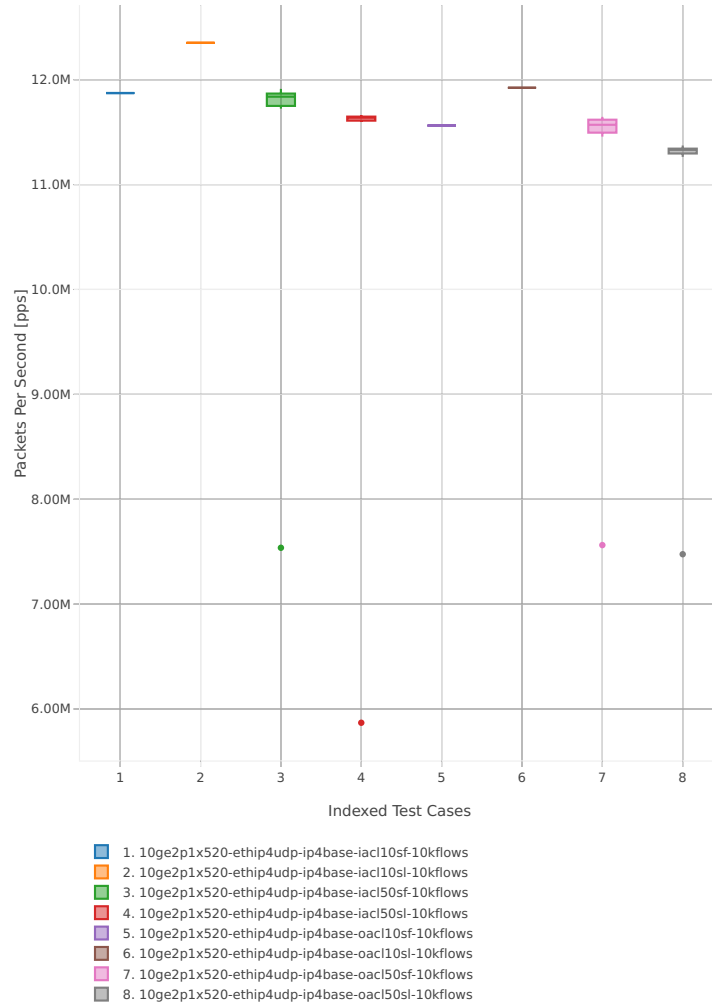


Throughput: ip4-3n-hsw-x520-64b-1t1c-features-pdr

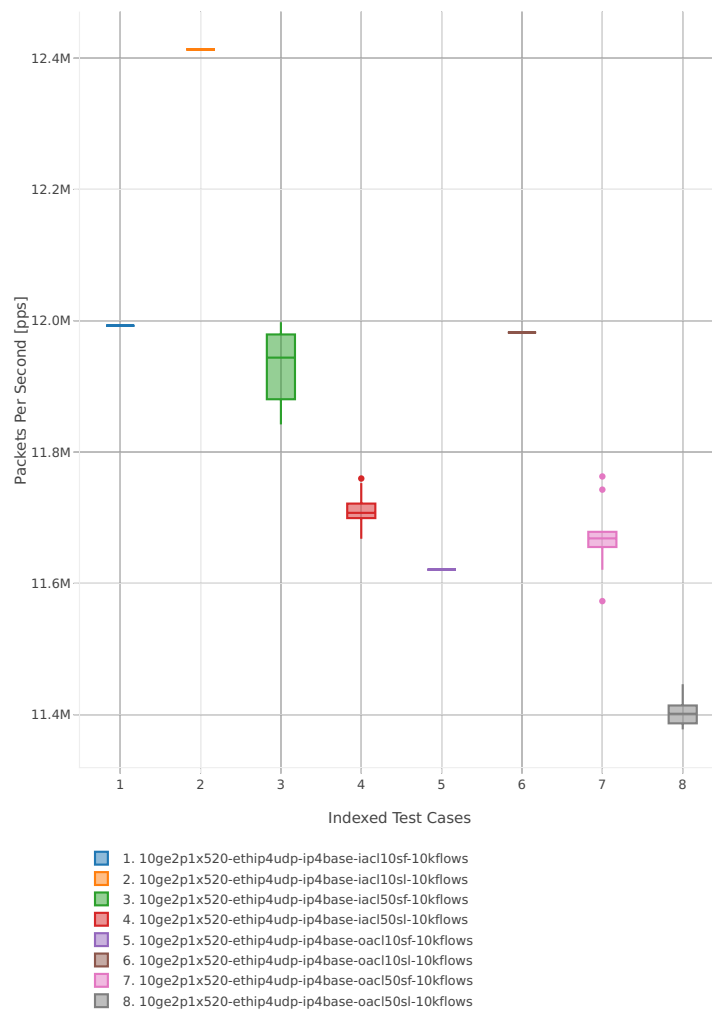


64b-2t2c-features

Throughput: ip4-3n-hsw-x520-64b-2t2c-features-ndr



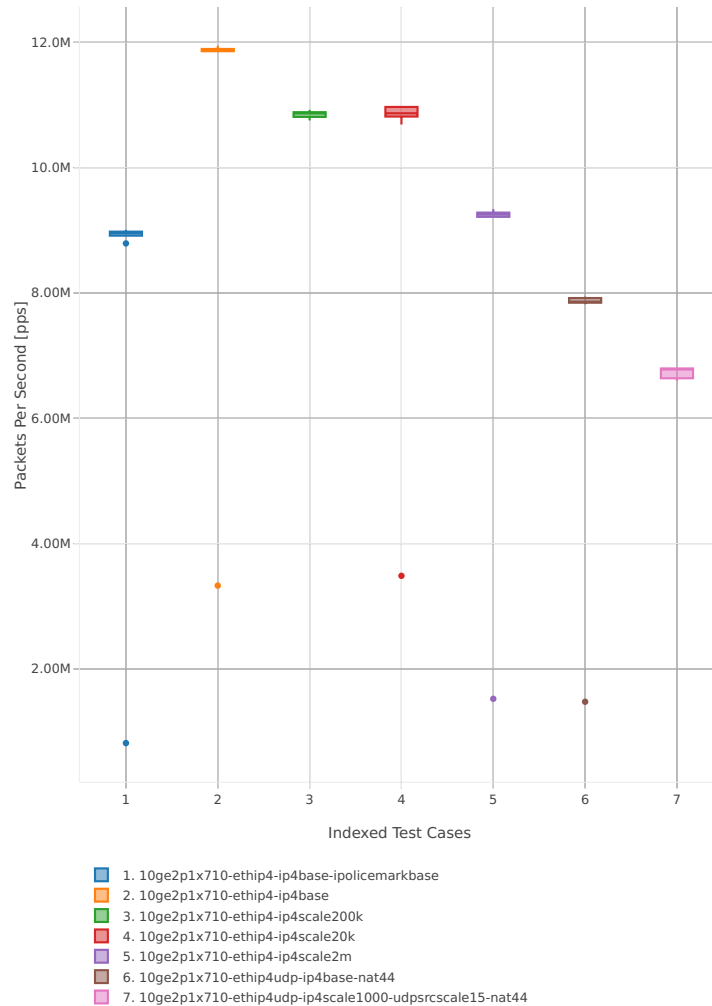
Throughput: ip4-3n-hsw-x520-64b-2t2c-features-pdr



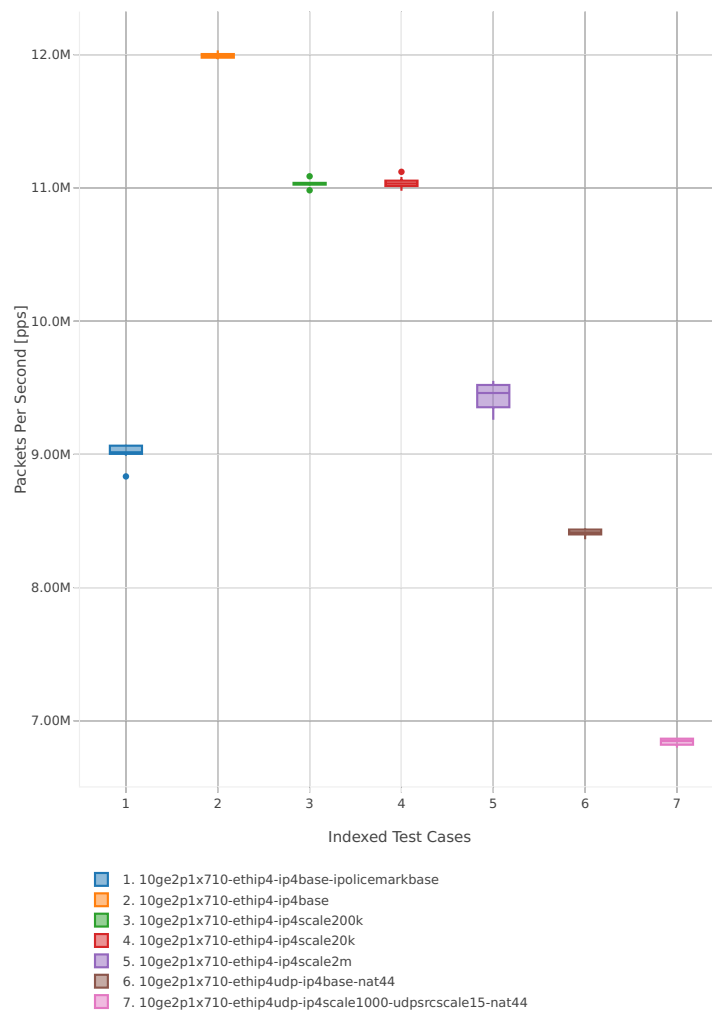
3n-hsw-x710

64b-1t1c-base_and_scale

Throughput: ip4-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

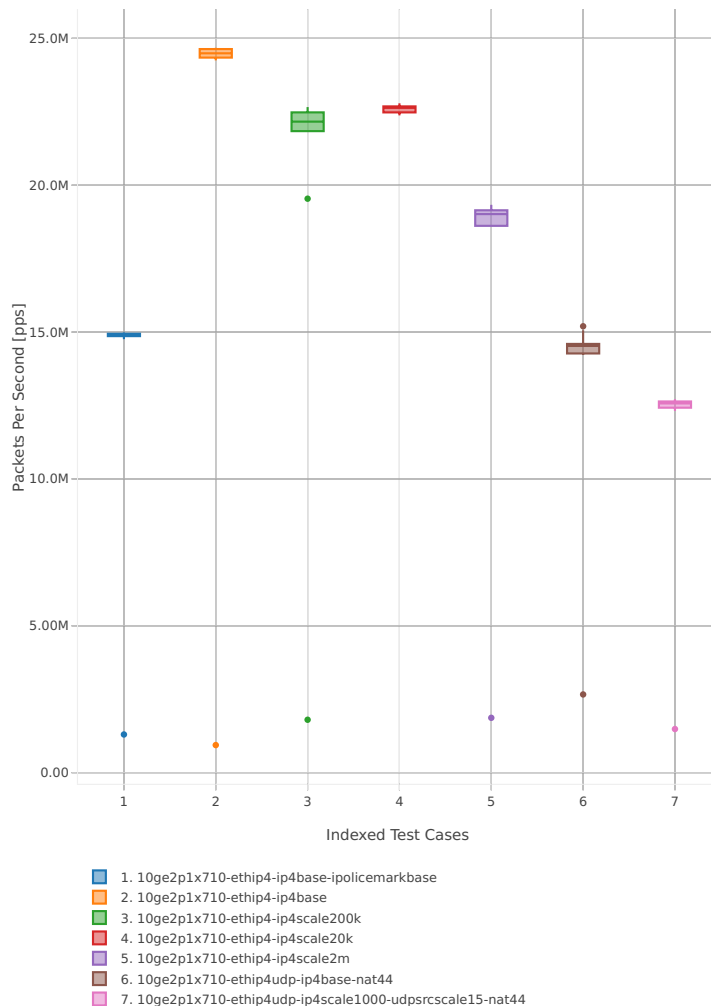


Throughput: ip4-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

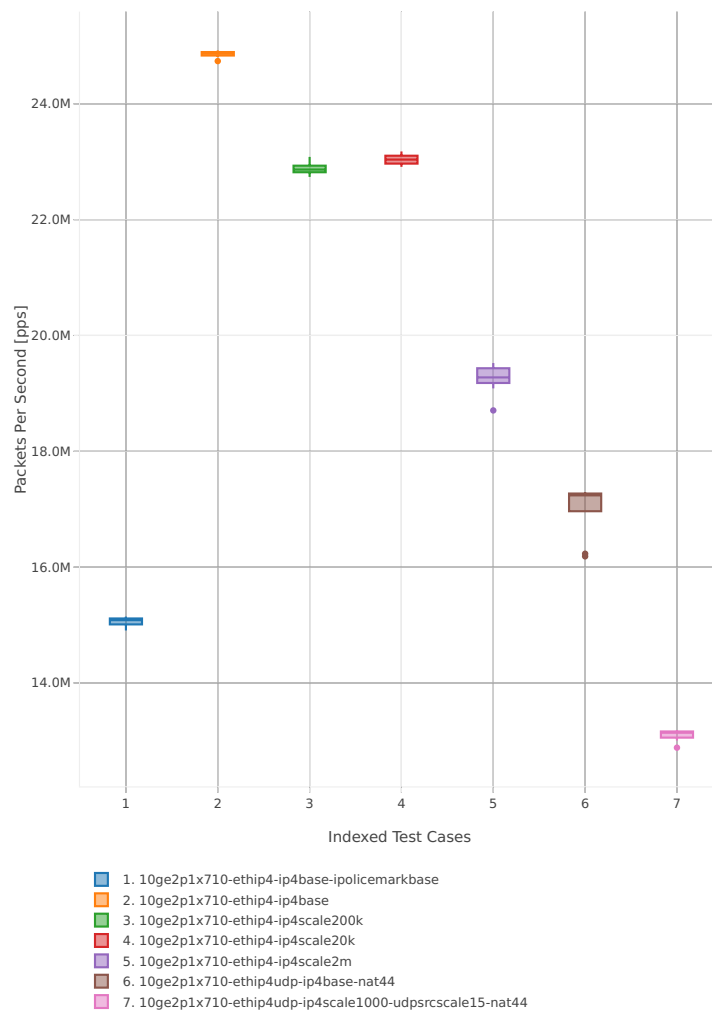


64b-2t2c-base_and_scale

Throughput: ip4-3n-hsw-x710-64b-2t2c-base_and_scale-ndr

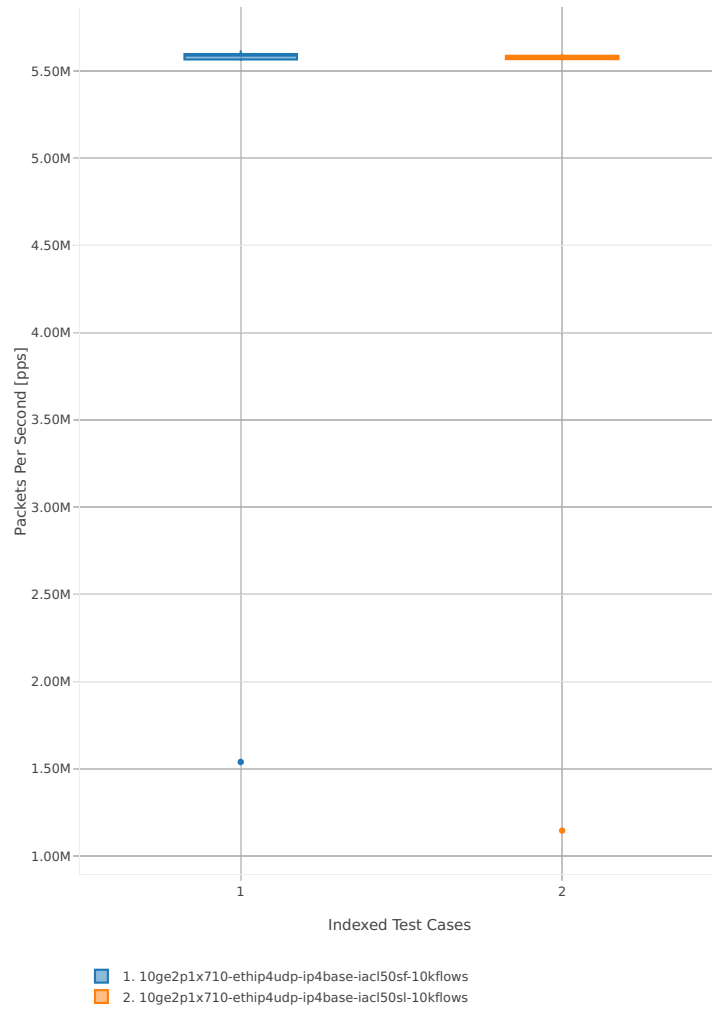


Throughput: ip4-3n-hsw-x710-64b-2t2c-base_and_scale-pdr

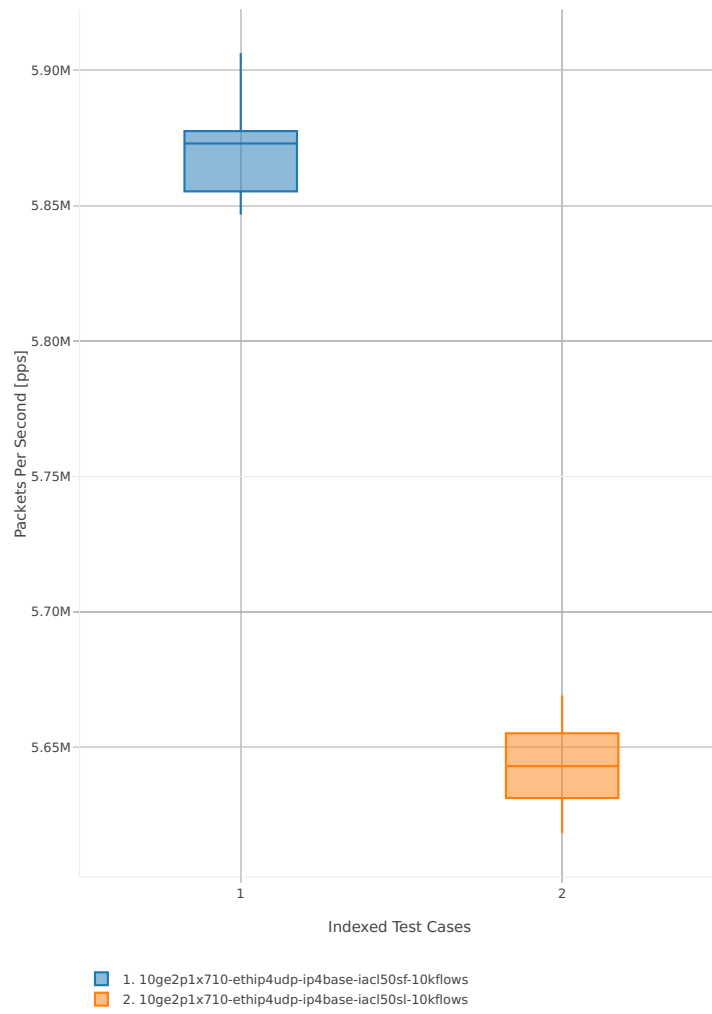


64b-1t1c-features

Throughput: ip4-3n-hsw-x710-64b-1t1c-features-ndr

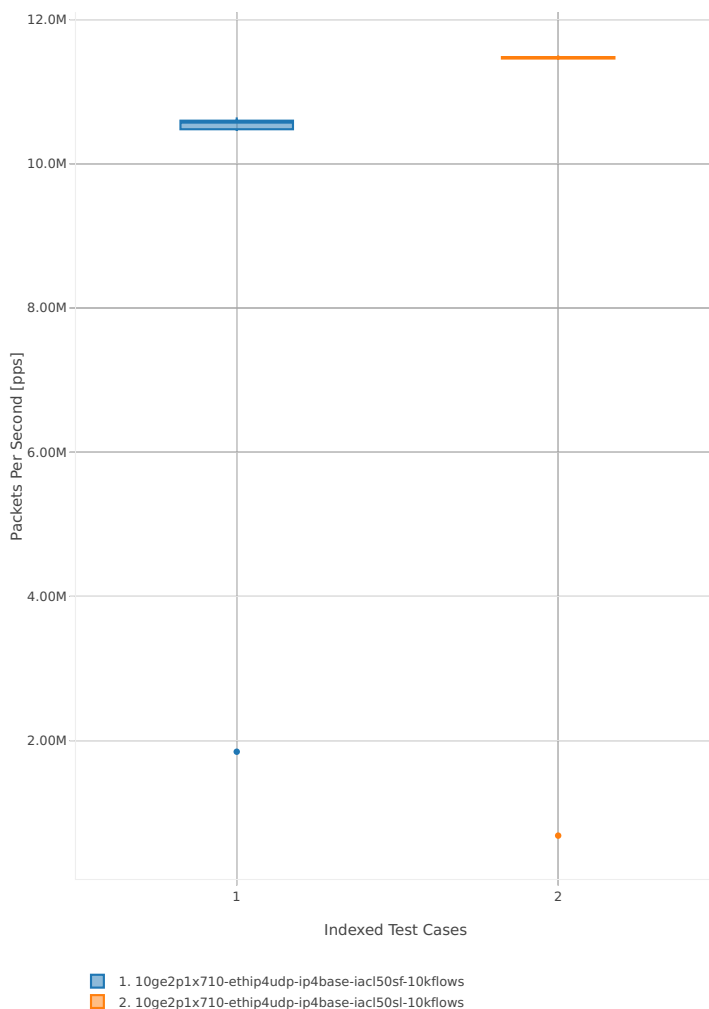


Throughput: ip4-3n-hsw-x710-64b-1t1c-features-pdr

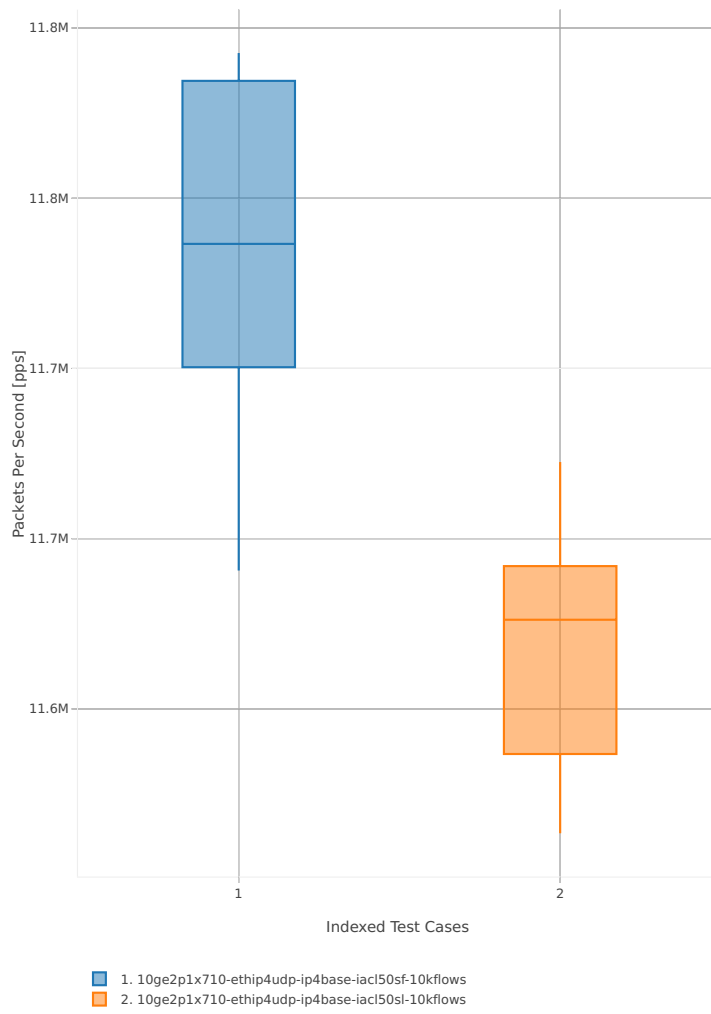


64b-2t2c-features

Throughput: ip4-3n-hsw-x710-64b-2t2c-features-ndr



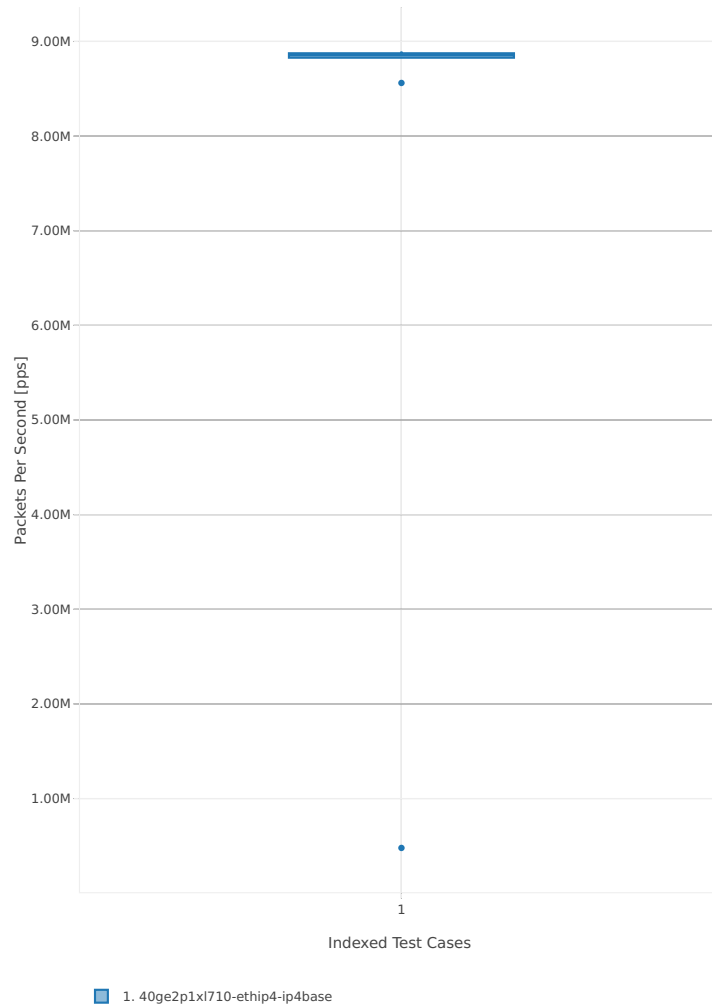
Throughput: ip4-3n-hsw-x710-64b-2t2c-features-pdr



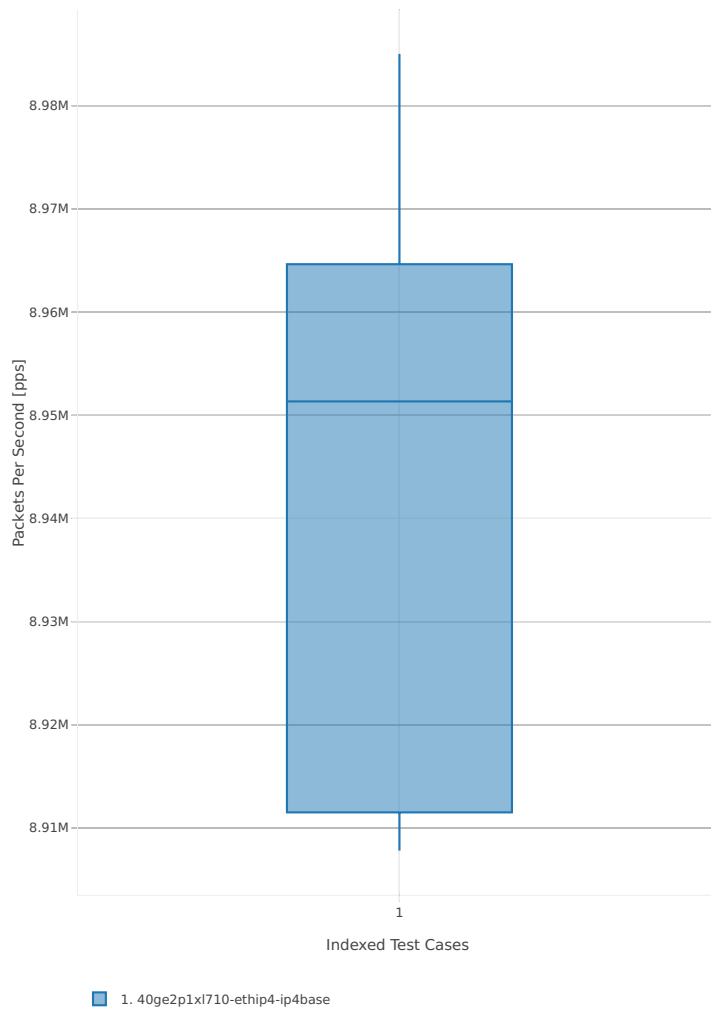
3n-hsw-xl710

64b-1t1c-base_and_scale

Throughput: ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr

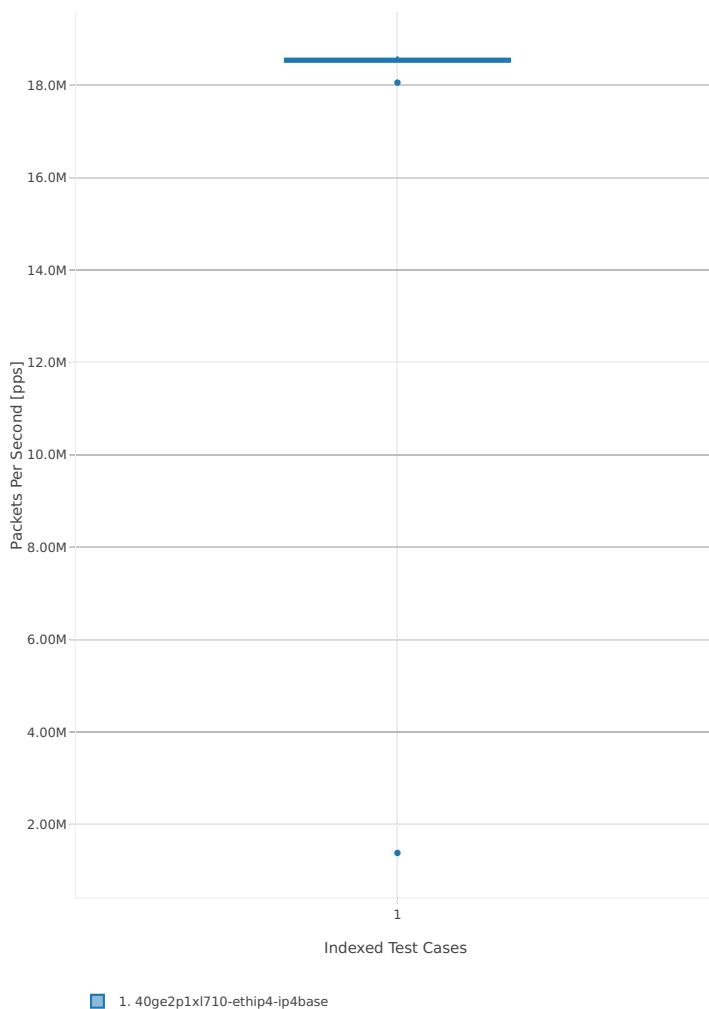


Throughput: ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

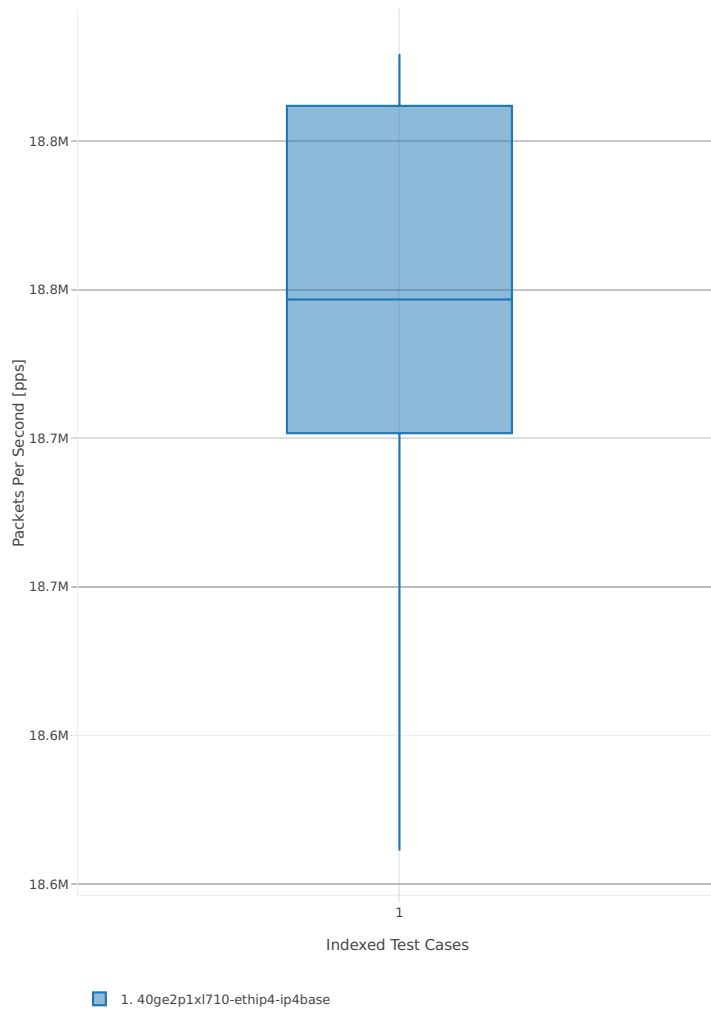


64b-2t2c-base_and_scale

Throughput: ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



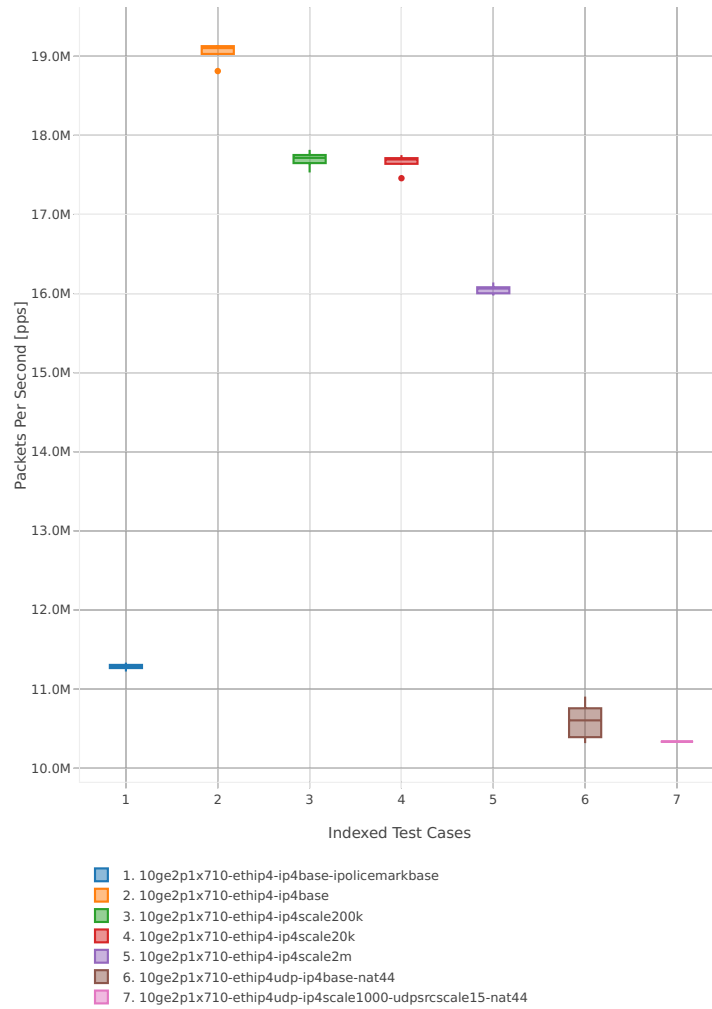
Throughput: ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr



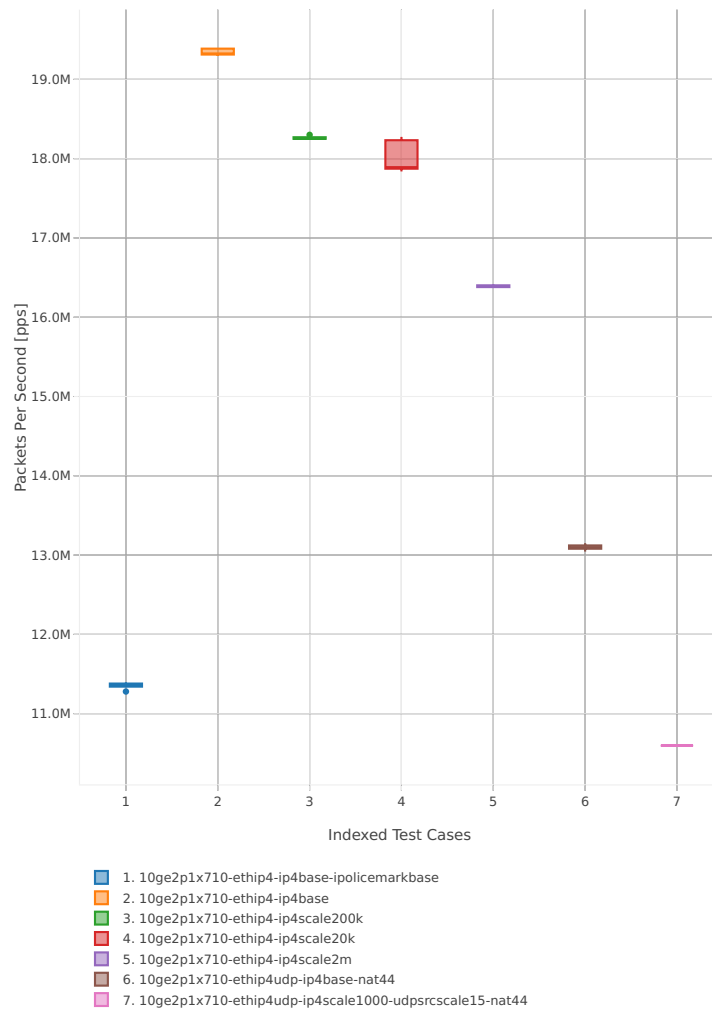
3n-skx-x710

64b-2t1c-base_and_scale

Throughput: ip4-3n-skx-x710-64b-2t1c-base_and_scale-ndr

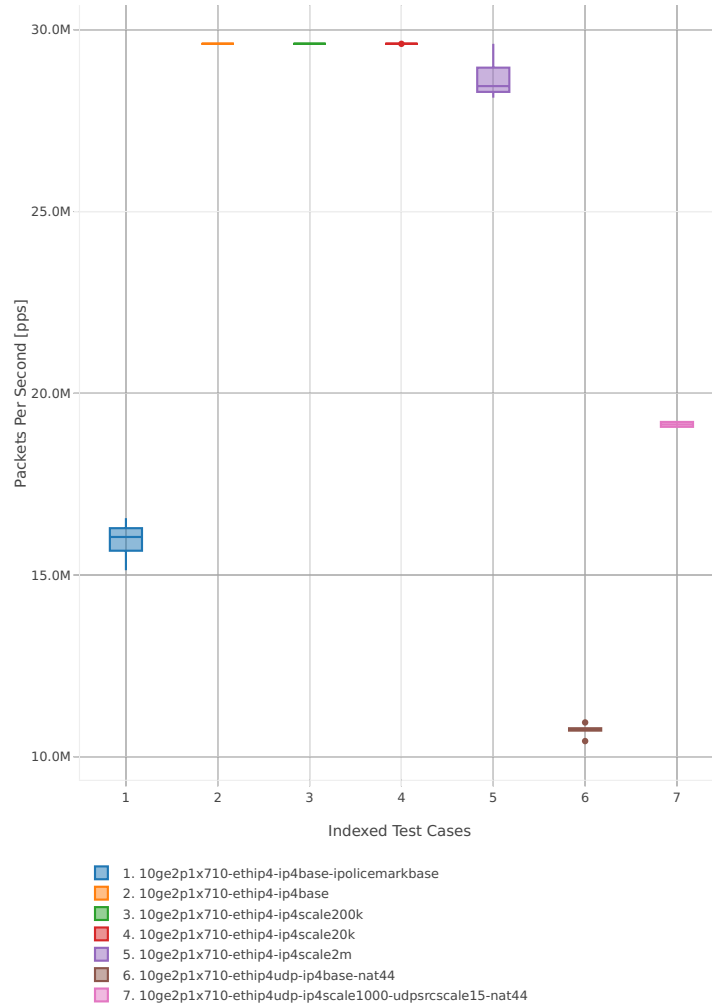


Throughput: ip4-3n-skx-x710-64b-2t1c-base_and_scale-pdr

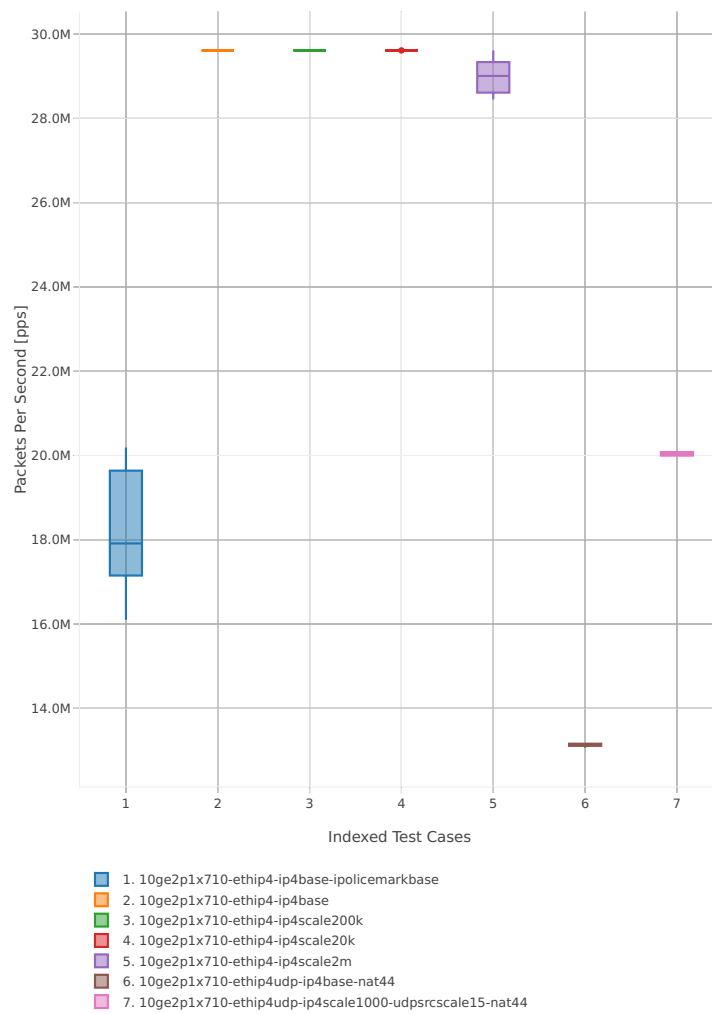


64b-4t2c-base_and_scale

Throughput: ip4-3n-skx-x710-64b-4t2c-base_and_scale-ndr

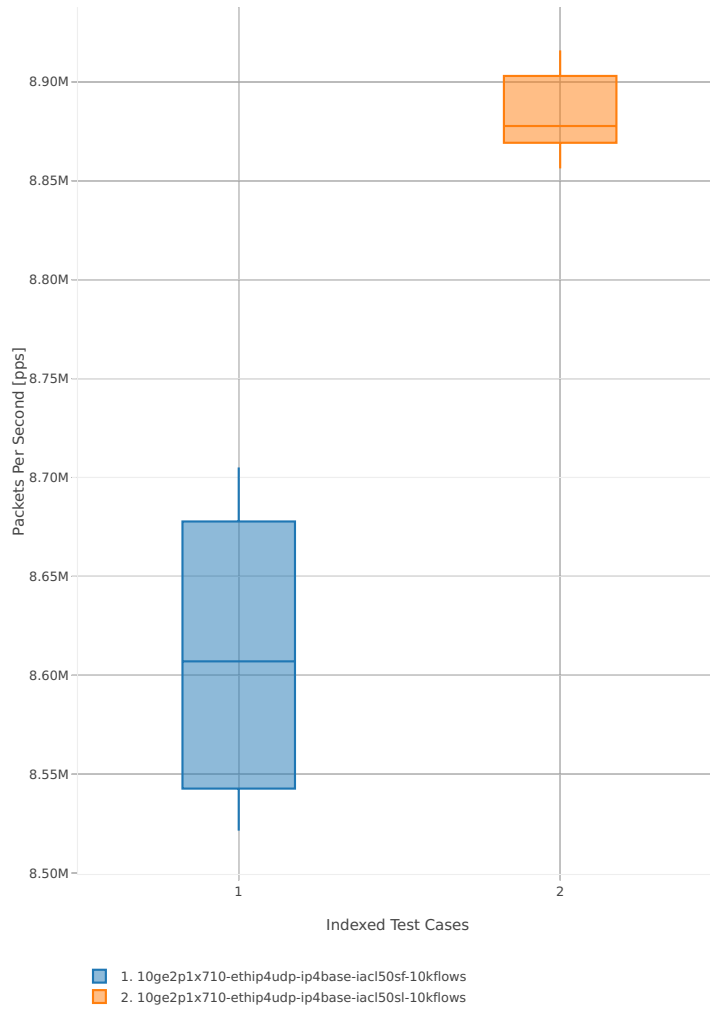


Throughput: ip4-3n-skx-x710-64b-4t2c-base_and_scale-pdr

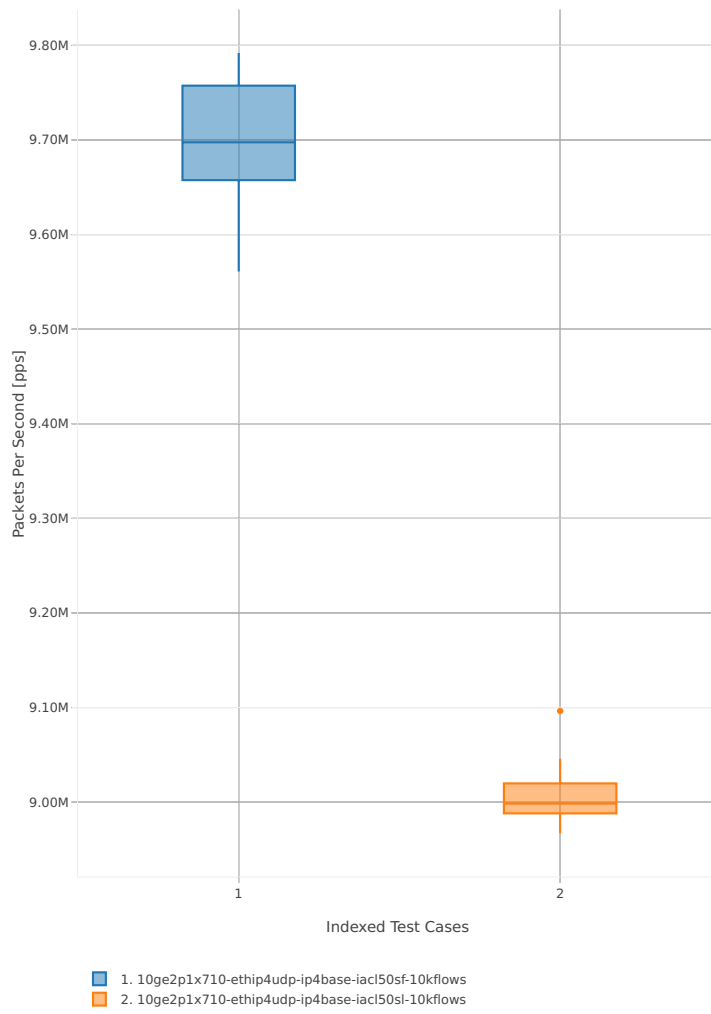


64b-2t1c-features

Throughput: ip4-3n-skx-x710-64b-2t1c-features-ndr

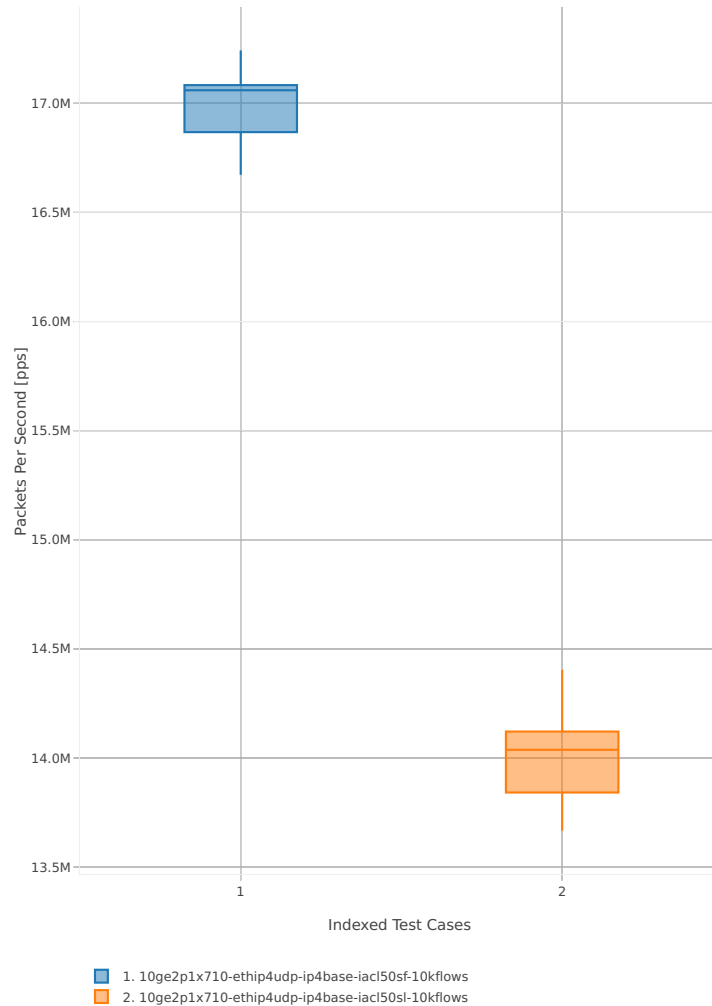


Throughput: ip4-3n-skx-x710-64b-2t1c-features-pdr

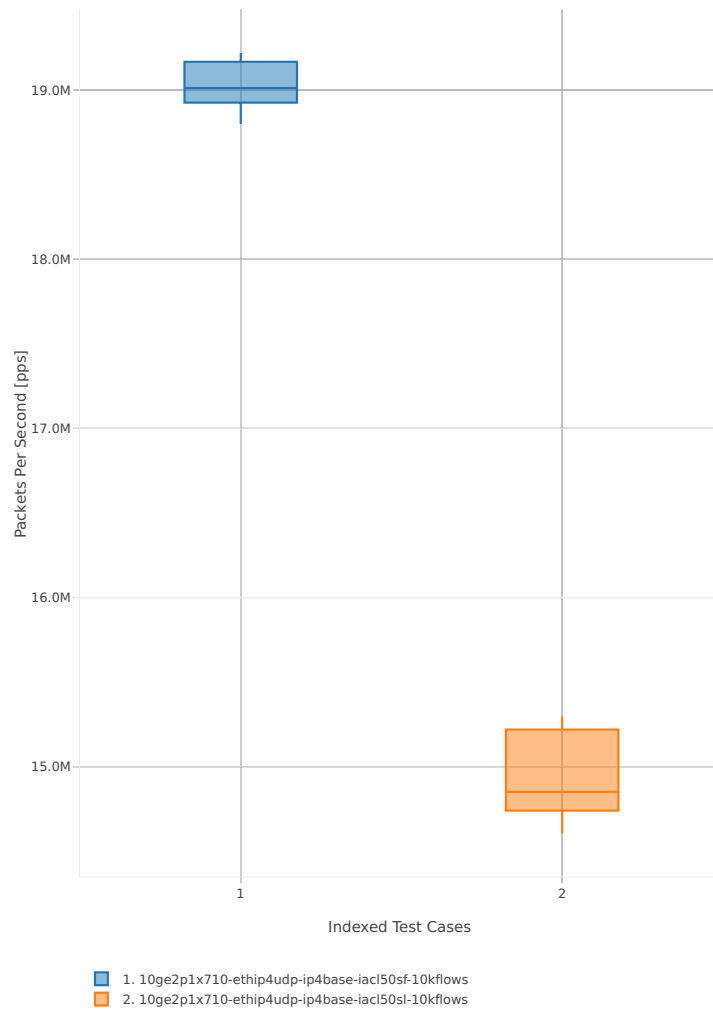


64b-4t2c-features

Throughput: ip4-3n-skx-x710-64b-4t2c-features-ndr



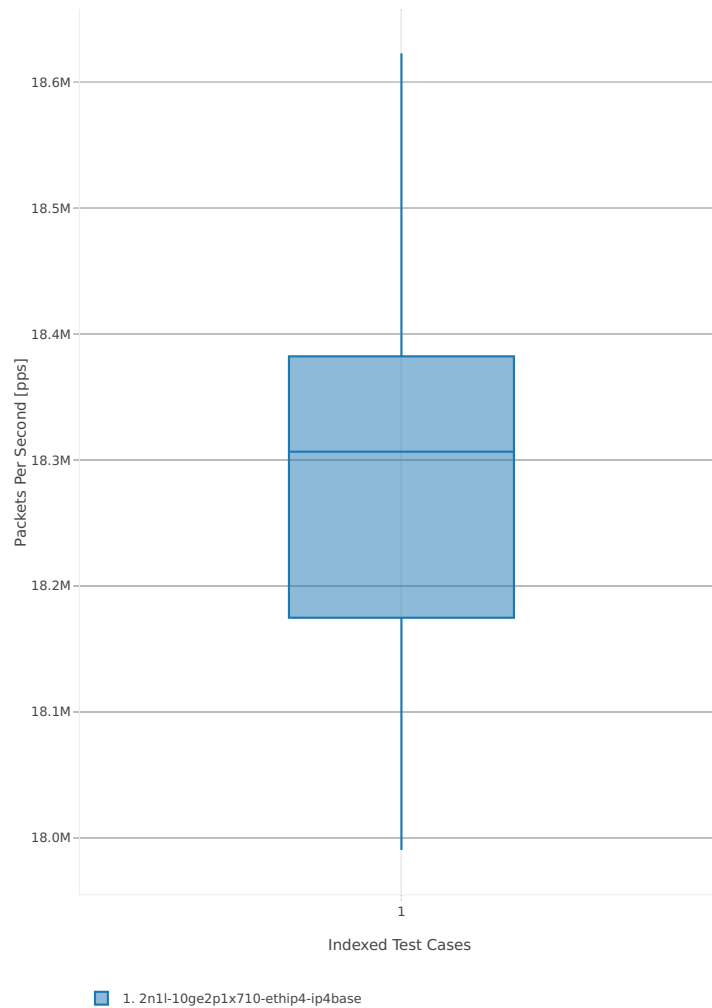
Throughput: ip4-3n-skx-x710-64b-4t2c-features-pdr



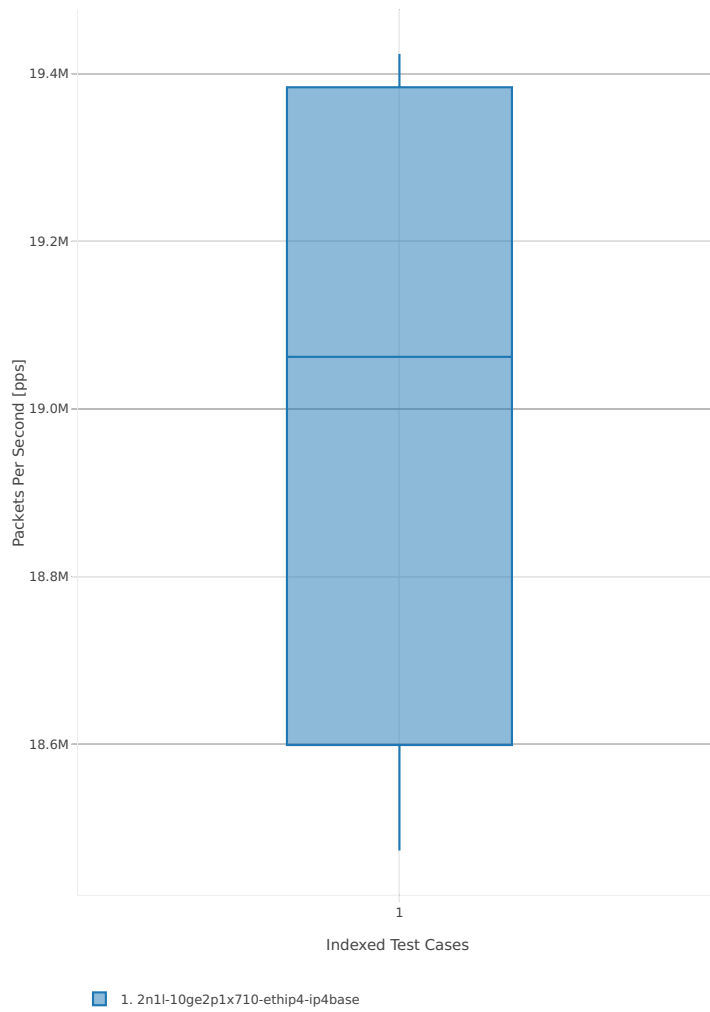
2n-skx-x710

64b-2t1c-base_and_scale

Throughput: ip4-2n-skx-x710-64b-2t1c-base_and_scale-ndr

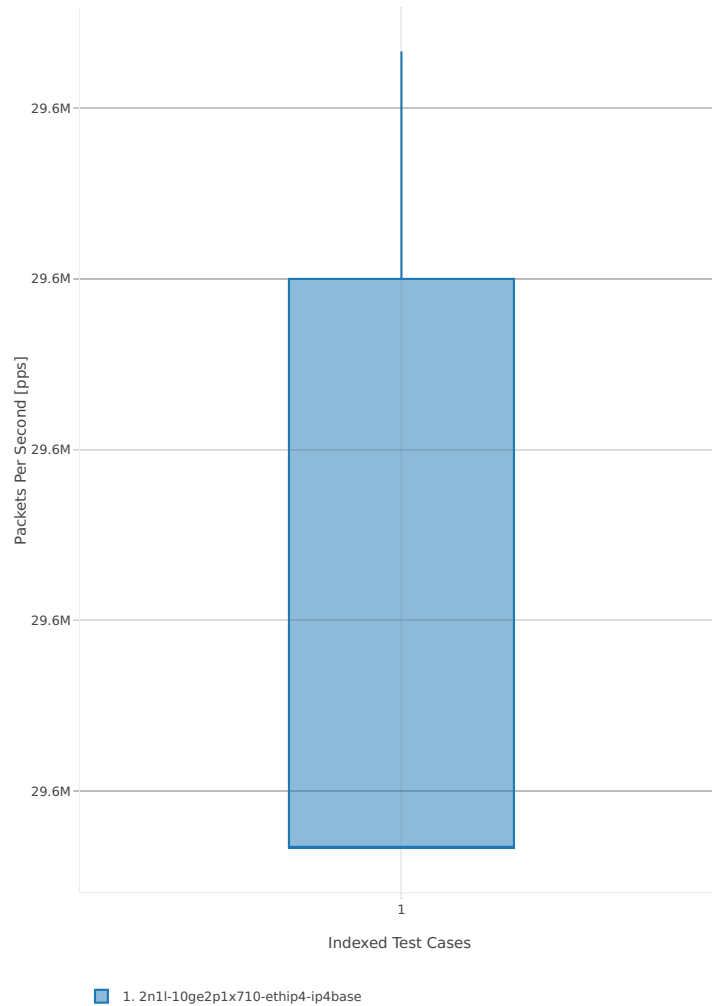


Throughput: ip4-2n-skx-x710-64b-2t1c-base_and_scale-pdr

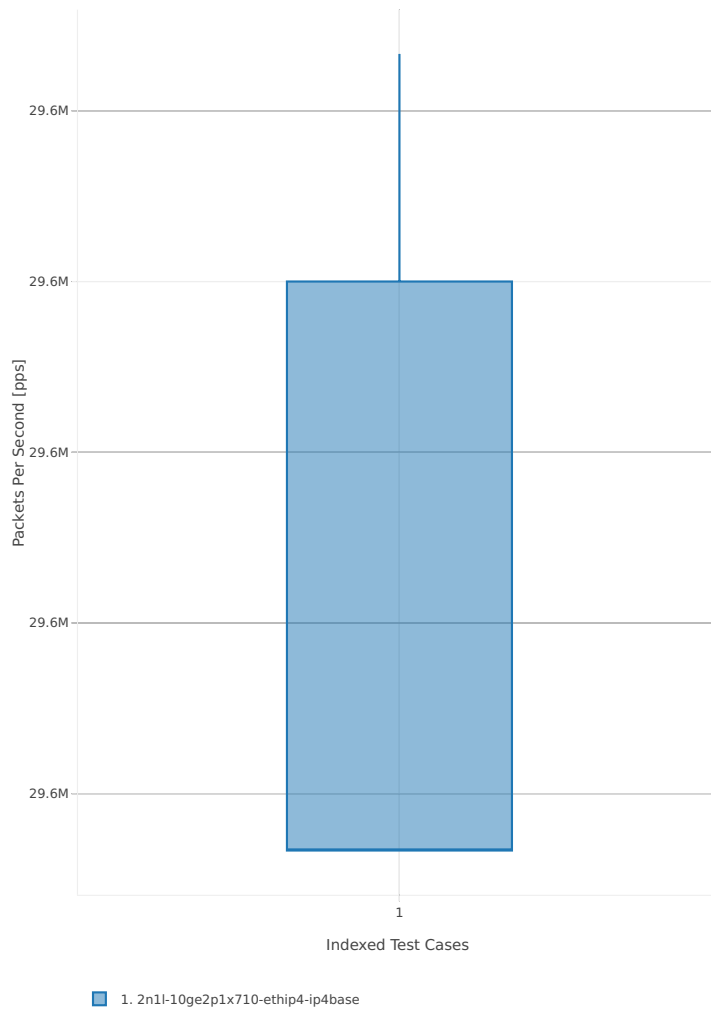


64b-4t2c-base_and_scale

Throughput: ip4-2n-skx-x710-64b-4t2c-base_and_scale-ndr



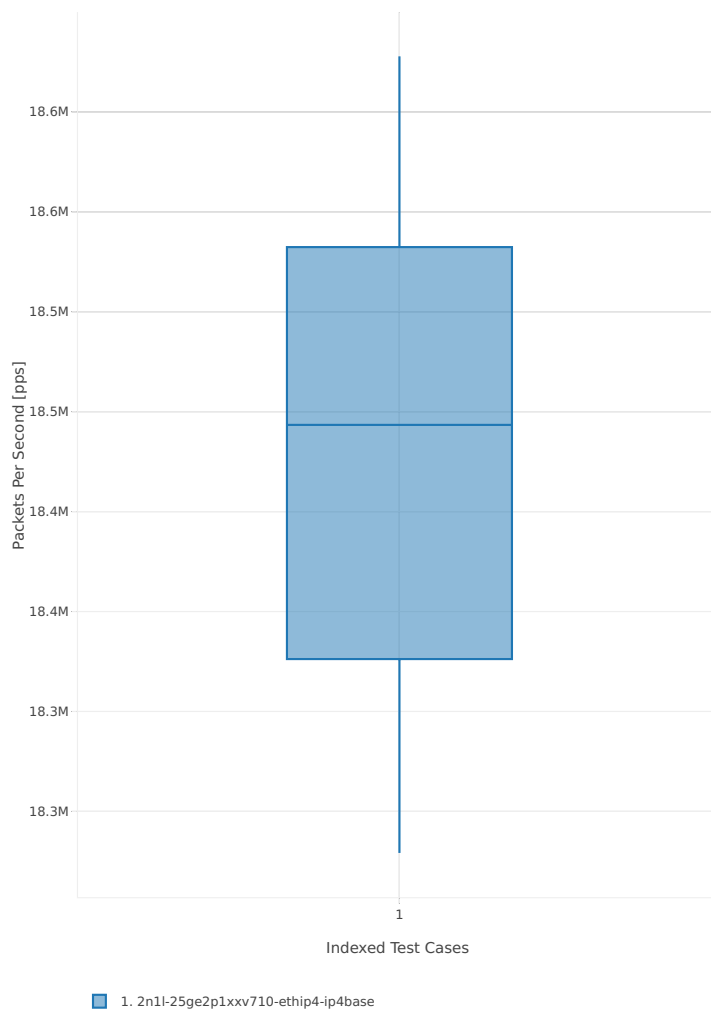
Throughput: ip4-2n-skx-x710-64b-4t2c-base_and_scale-pdr



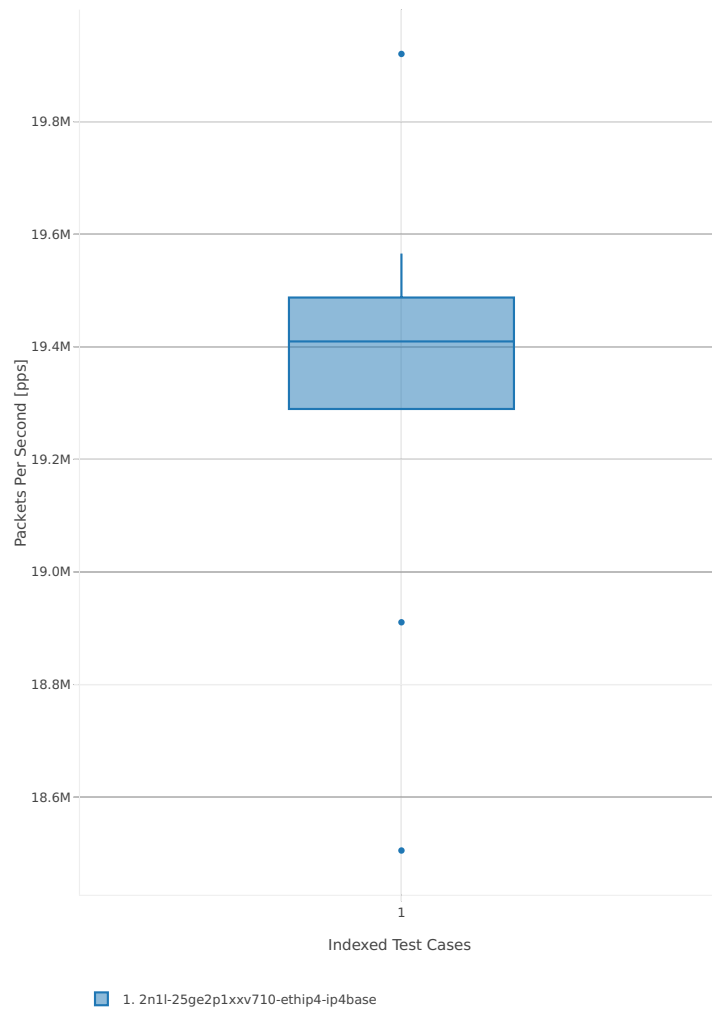
2n-skx-xxv710

64b-2t1c-base_and_scale

Throughput: ip4-2n-skx-xxv710-64b-2t1c-base_and_scale-ndr

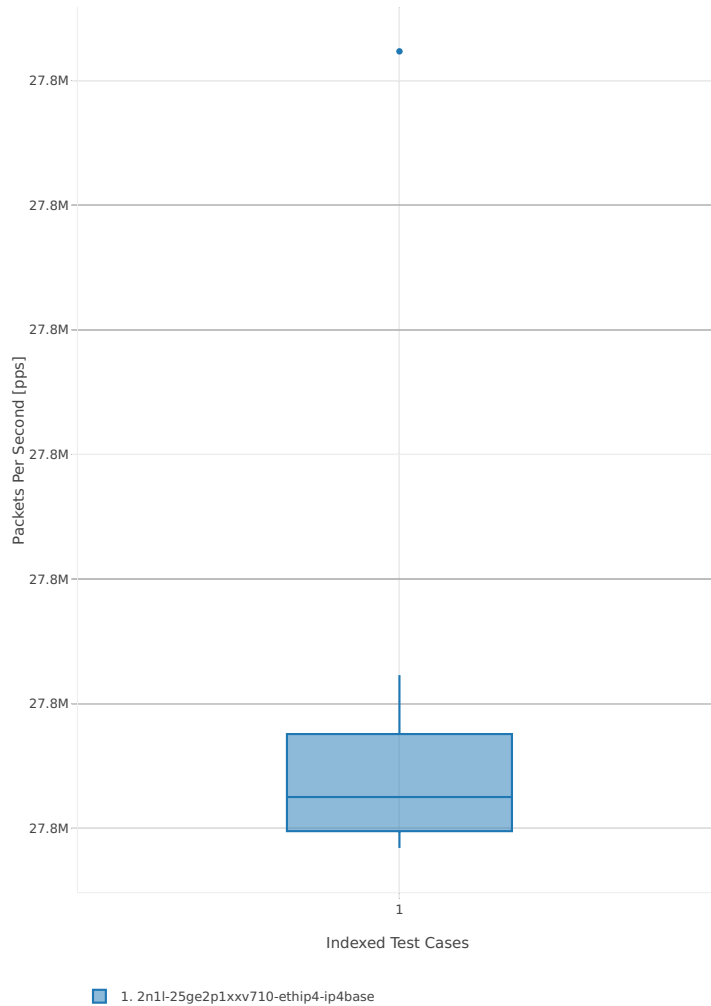


Throughput: ip4-2n-skx-xxv710-64b-2t1c-base_and_scale-pdr

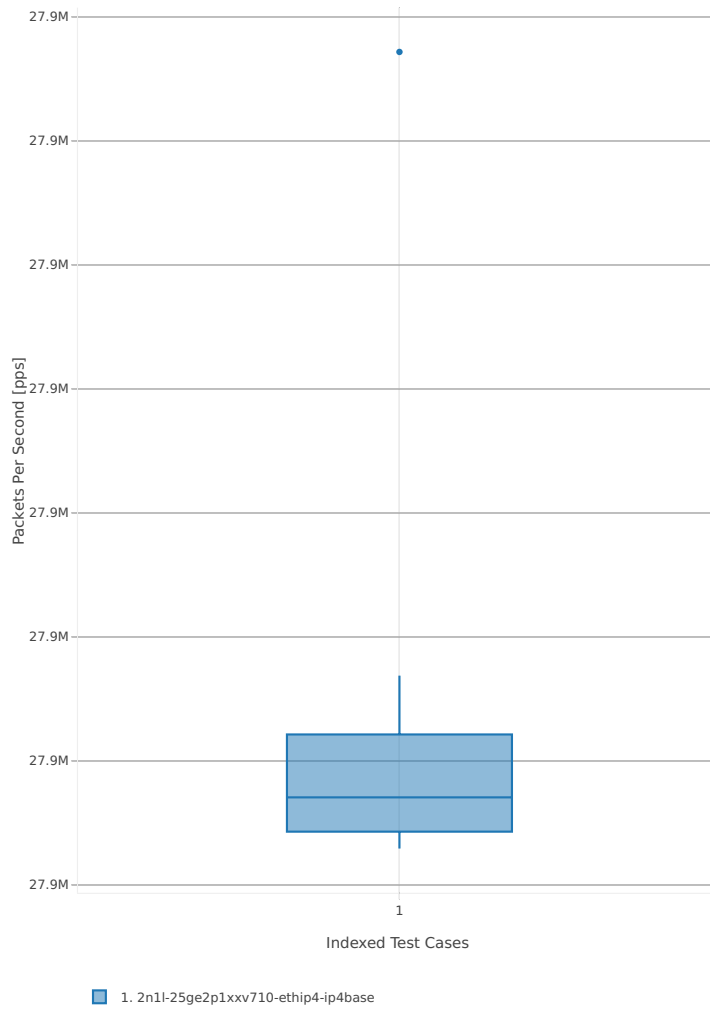


64b-4t2c-base_and_scale

Throughput: ip4-2n-skx-xxv710-64b-4t2c-base_and_scale-ndr



Throughput: ip4-2n-skx-xxv710-64b-4t2c-base_and_scale-pdr



2.3.3 IPv6 Routing

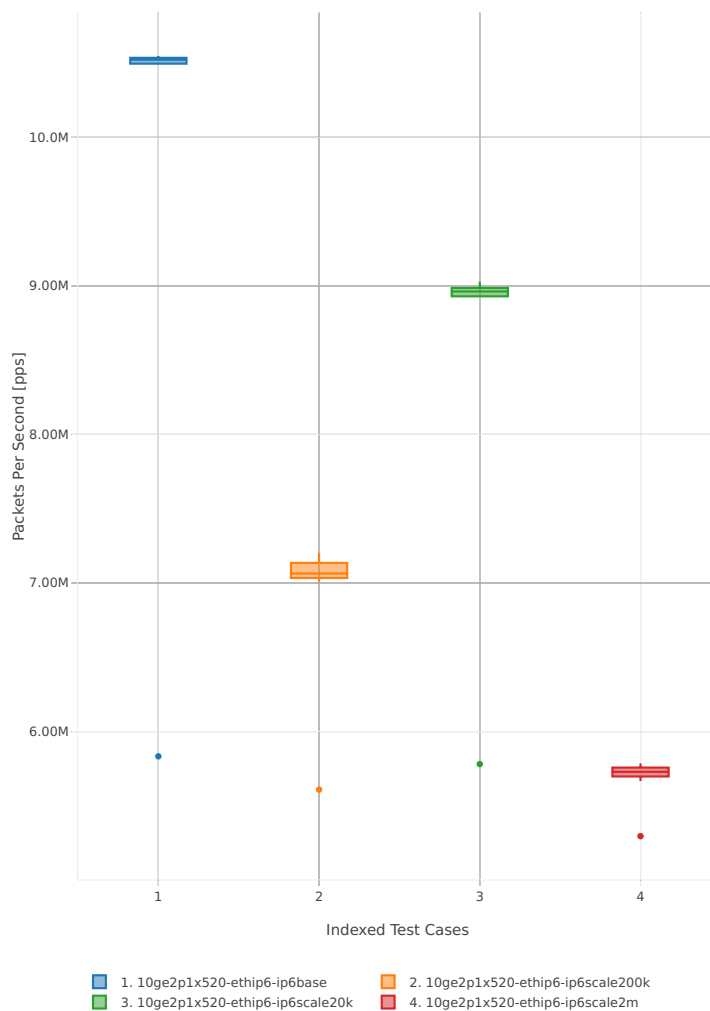
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv6 Routed-Forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)³¹.

3n-hsw-x520

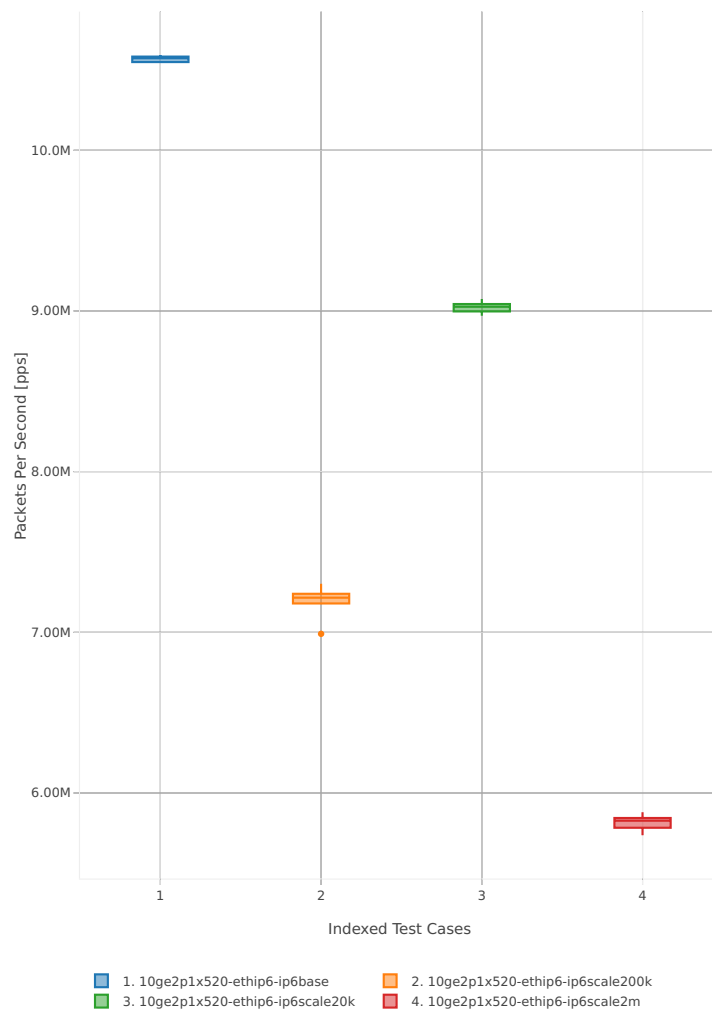
78b-1t1c-base_and_scale

Throughput: ip6-3n-hsw-x520-78b-1t1c-base_and_scale-ndr



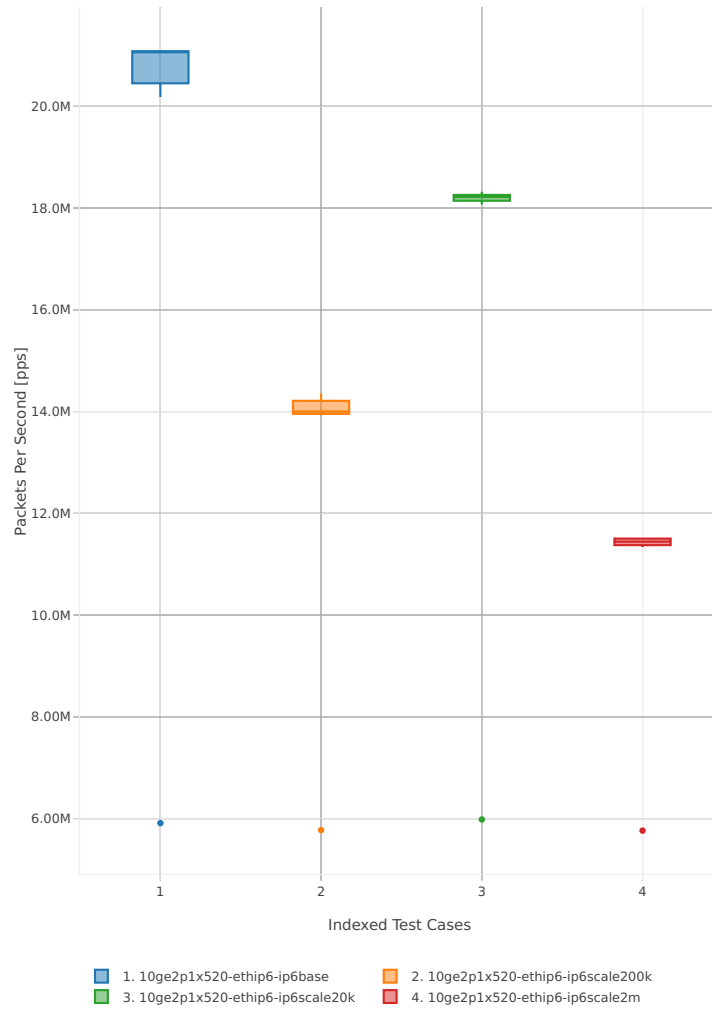
³¹ <https://git.fd.io/csit/tree/tests/vpp/perf/ip6?h=rls1807>

Throughput: ip6-3n-hsw-x520-78b-1t1c-base_and_scale-pdr

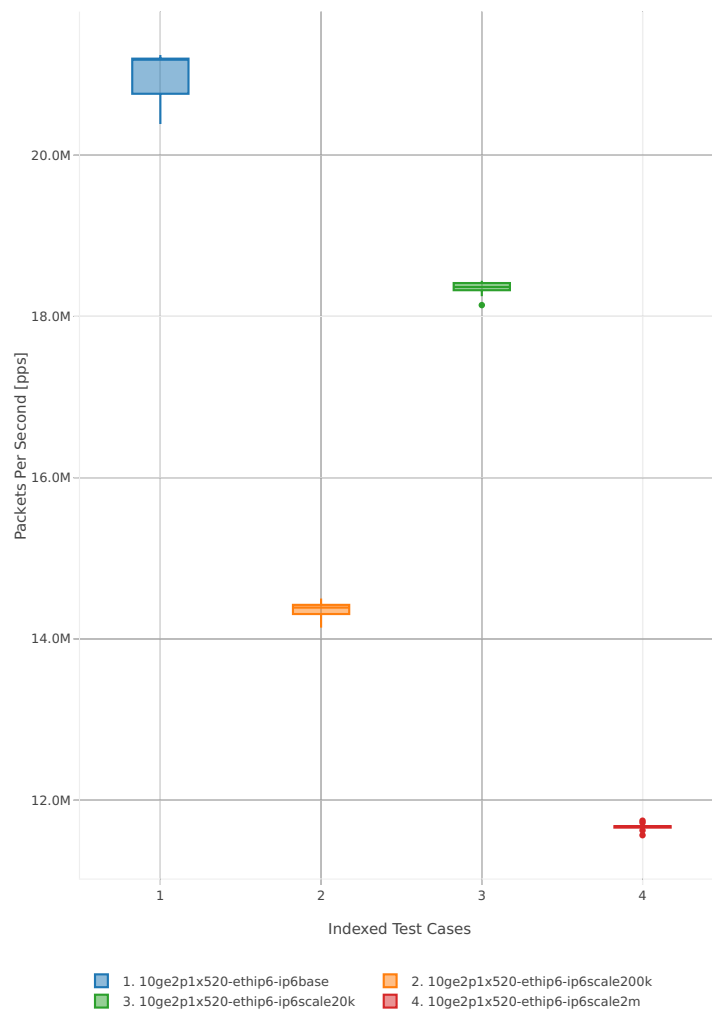


78b-2t2c-base_and_scale

Throughput: ip6-3n-hsw-x520-78b-2t2c-base_and_scale-ndr



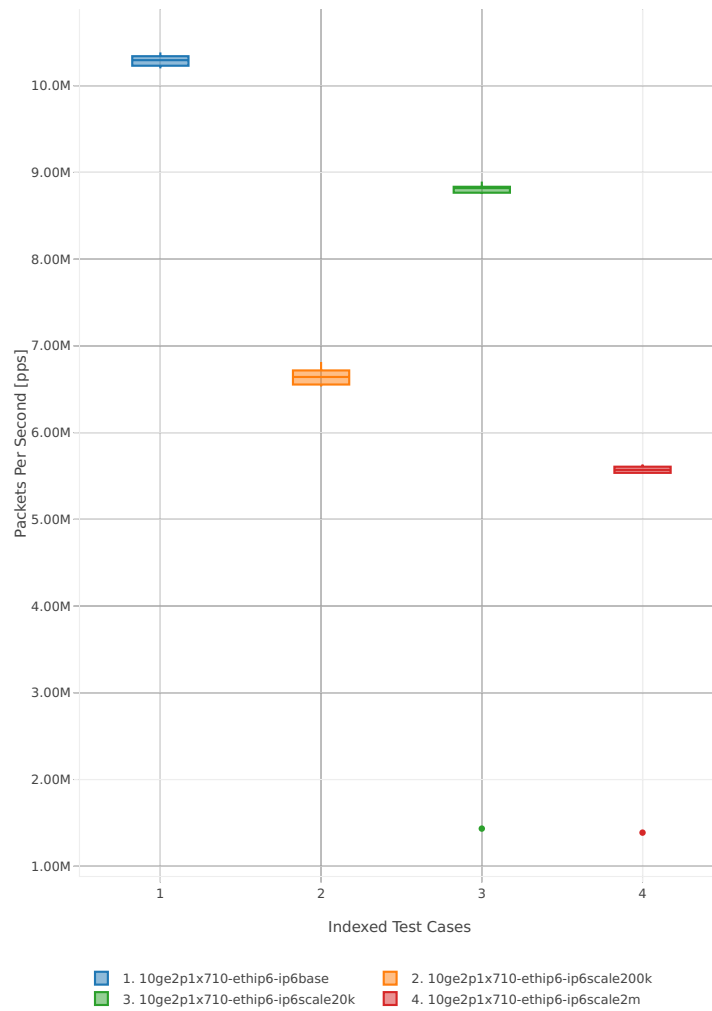
Throughput: ip6-3n-hsw-x520-78b-2t2c-base_and_scale-pdr



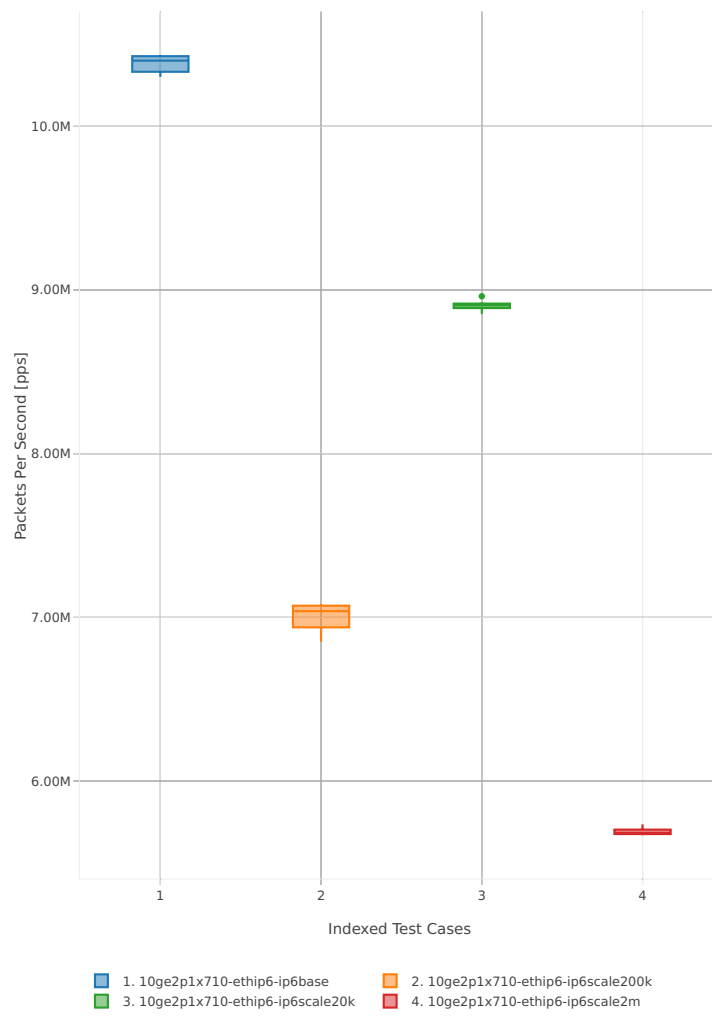
3n-hsw-x710

78b-1t1c-base_and_scale

Throughput: ip6-3n-hsw-x710-78b-1t1c-base_and_scale-ndr

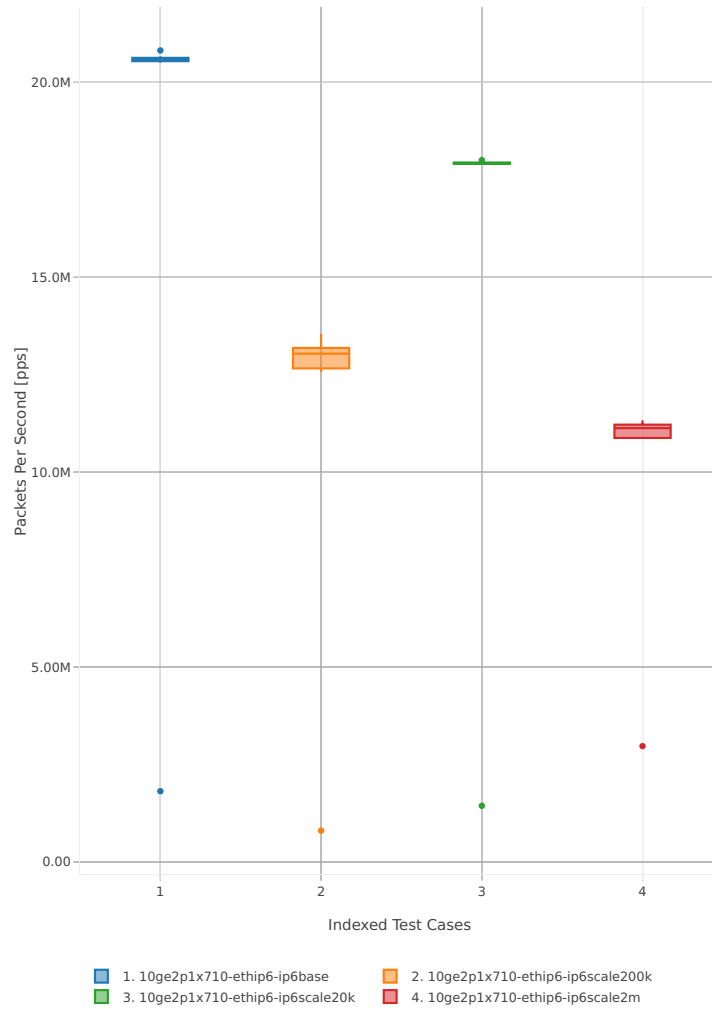


Throughput: ip6-3n-hsw-x710-78b-1t1c-base_and_scale-pdr

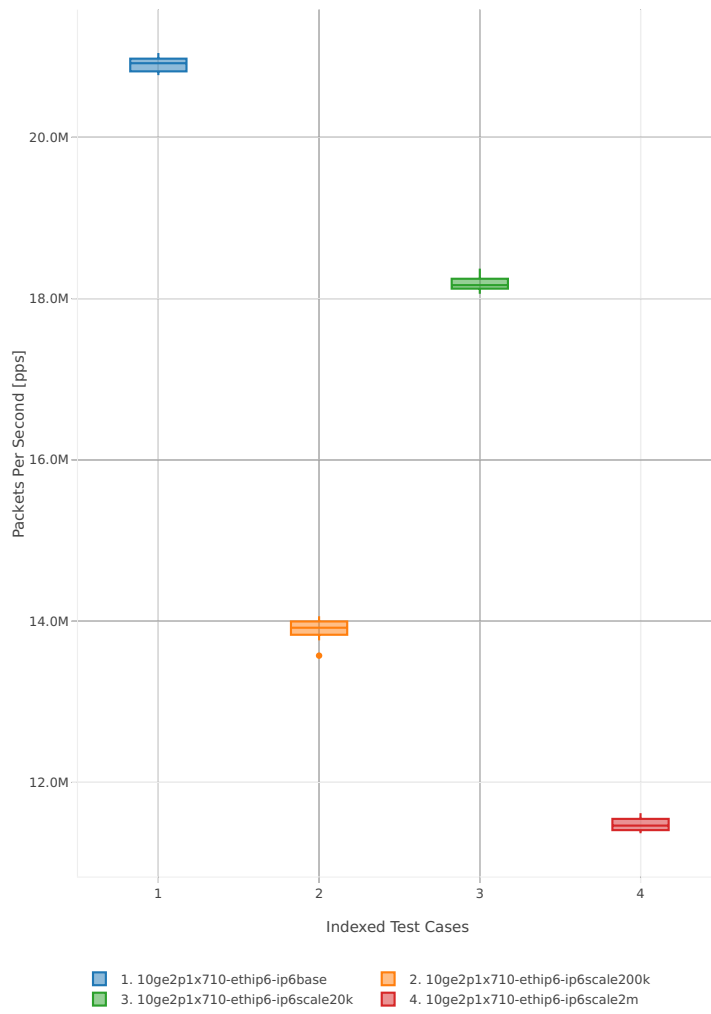


78b-2t2c-base_and_scale

Throughput: ip6-3n-hsw-x710-78b-2t2c-base_and_scale-ndr



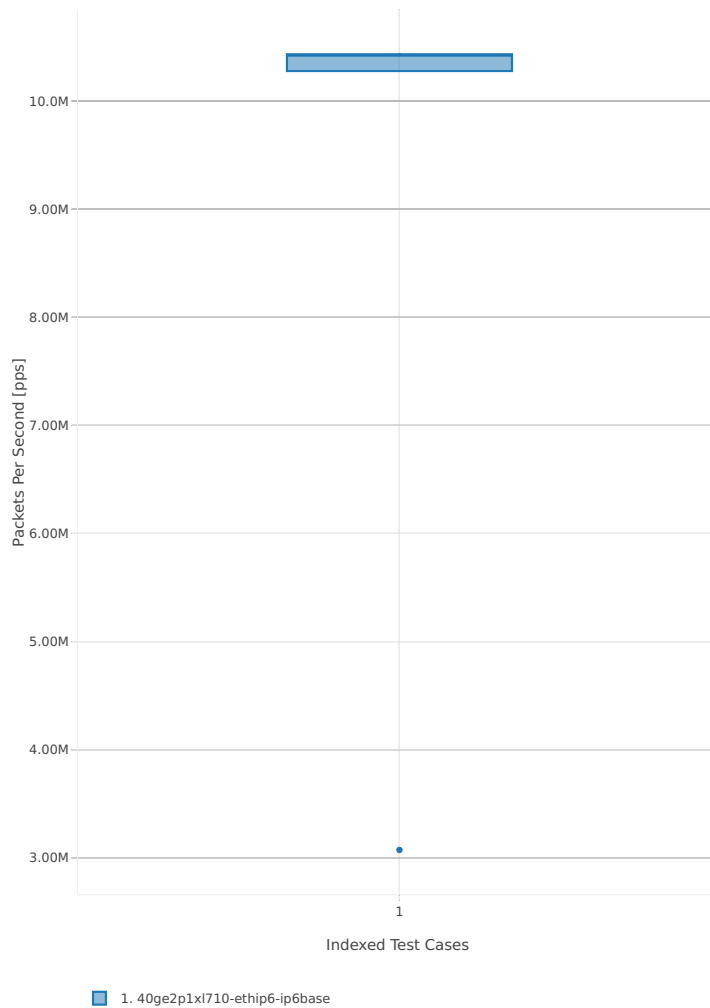
Throughput: ip6-3n-hsw-x710-78b-2t2c-base_and_scale-pdr



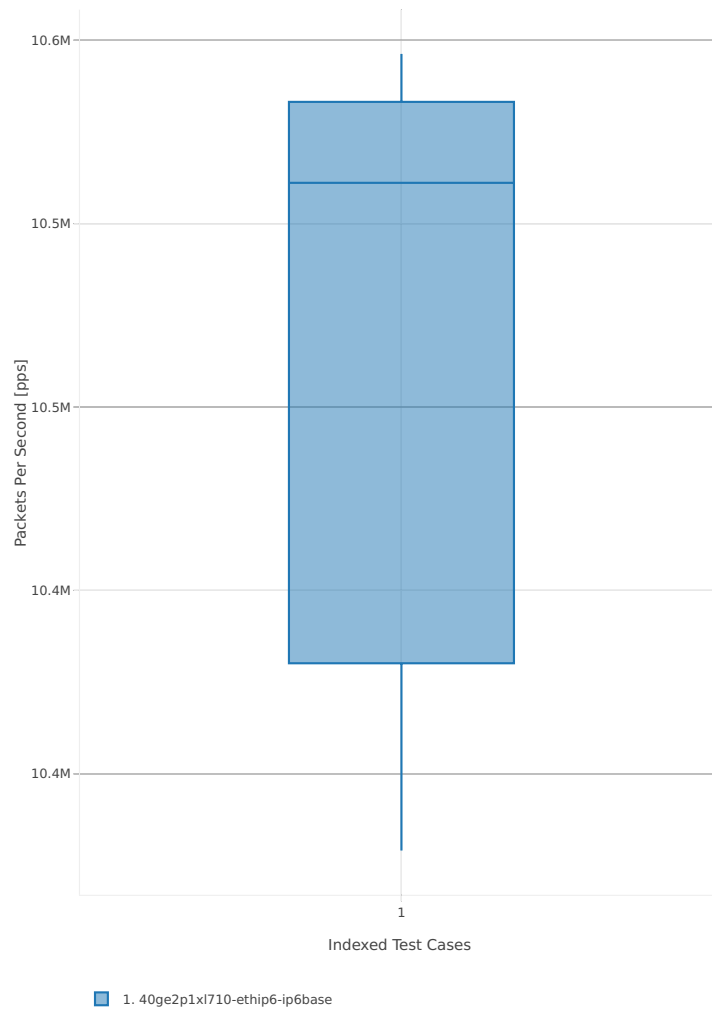
3n-hsw-xl710

78b-1t1c-base_and_scale

Throughput: ip6-3n-hsw-xl710-78b-1t1c-base_and_scale-ndr

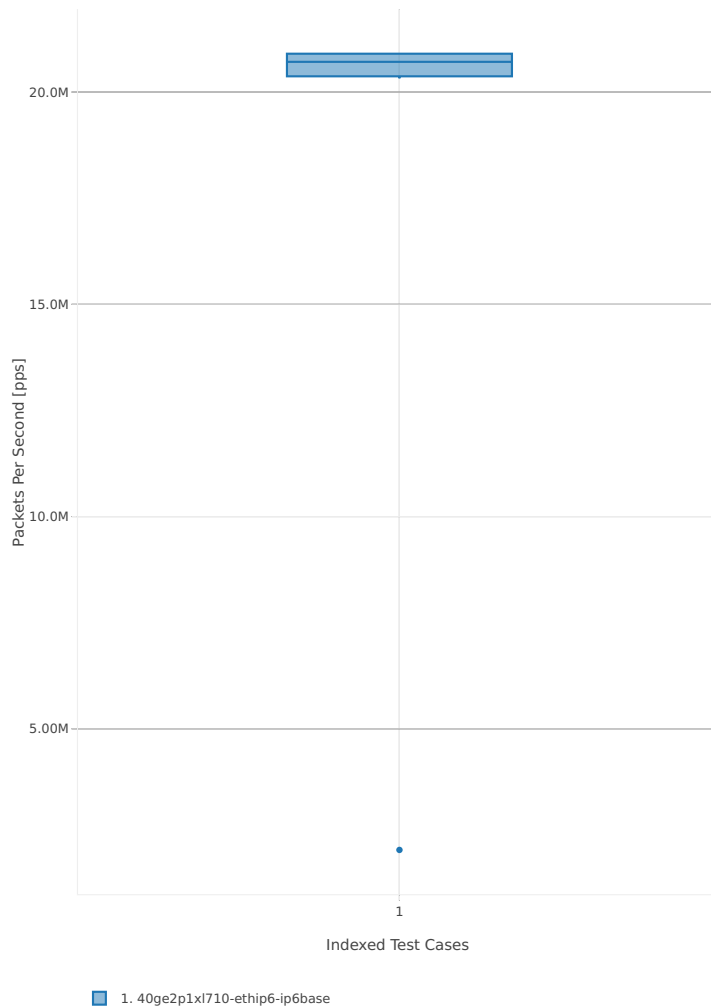


Throughput: ip6-3n-hsw-xl710-78b-1t1c-base_and_scale-pdr

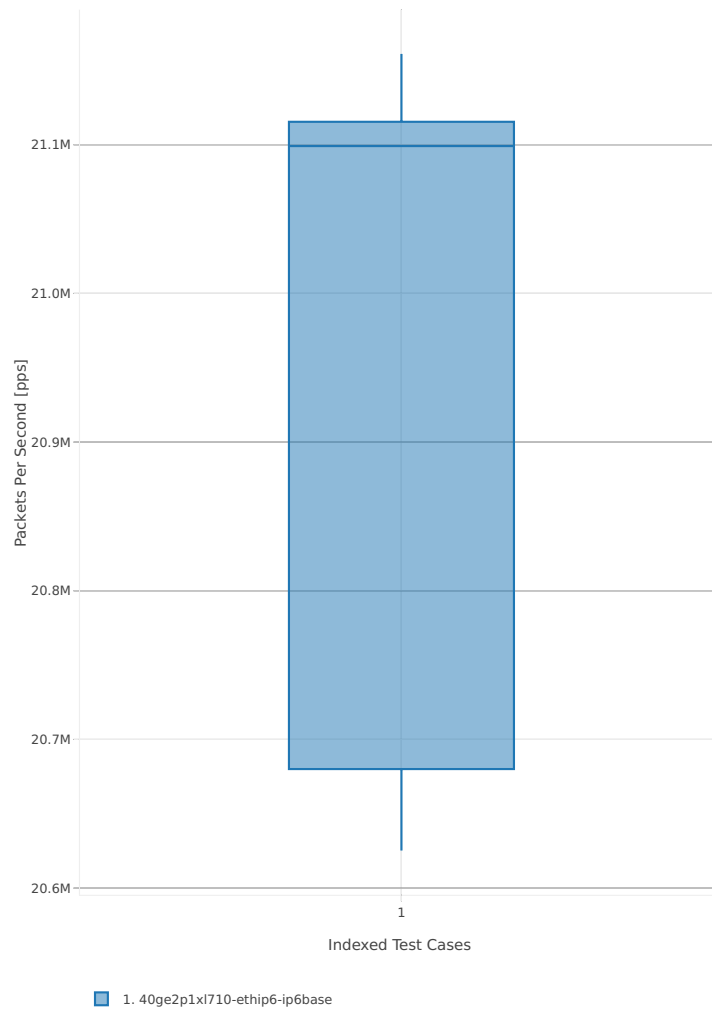


78b-2t2c-base_and_scale

Throughput: ip6-3n-hsw-xl710-78b-2t2c-base_and_scale-ndr



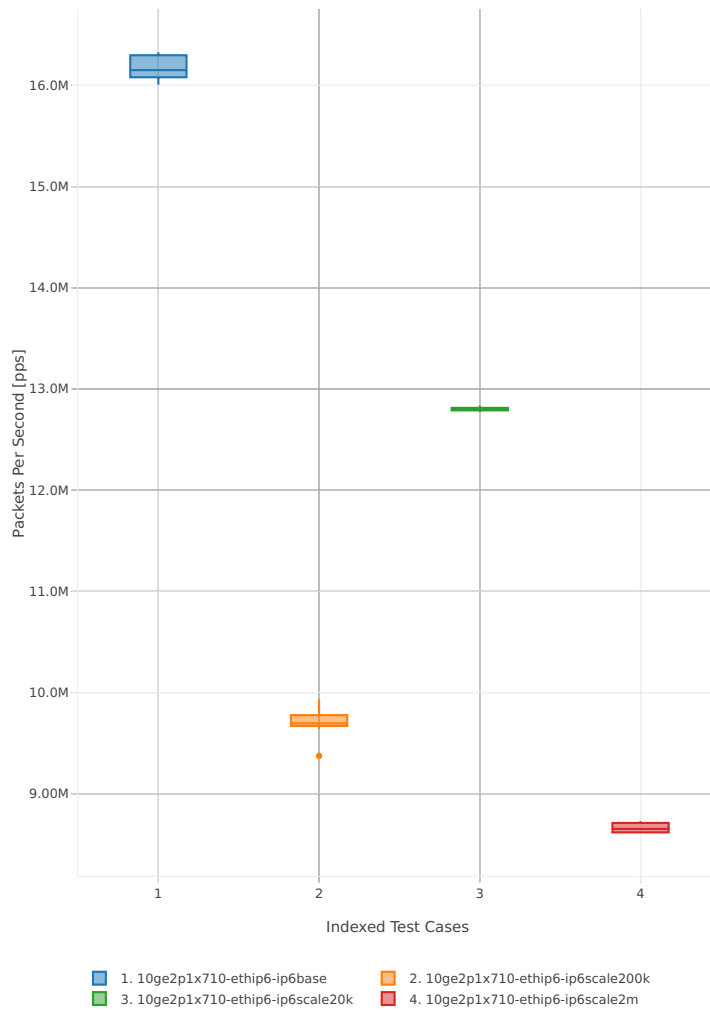
Throughput: ip6-3n-hsw-xl710-78b-2t2c-base_and_scale-pdr



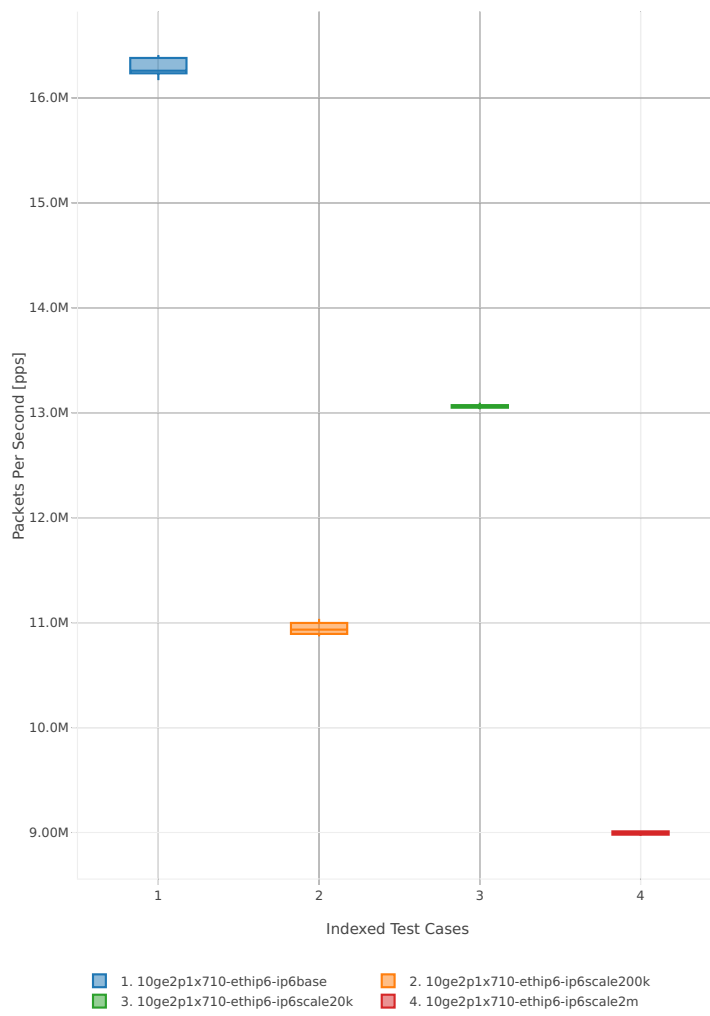
3n-skx-x710

78b-2t1c-base_and_scale

Throughput: ip6-3n-skx-x710-78b-2t1c-base_and_scale-ndr

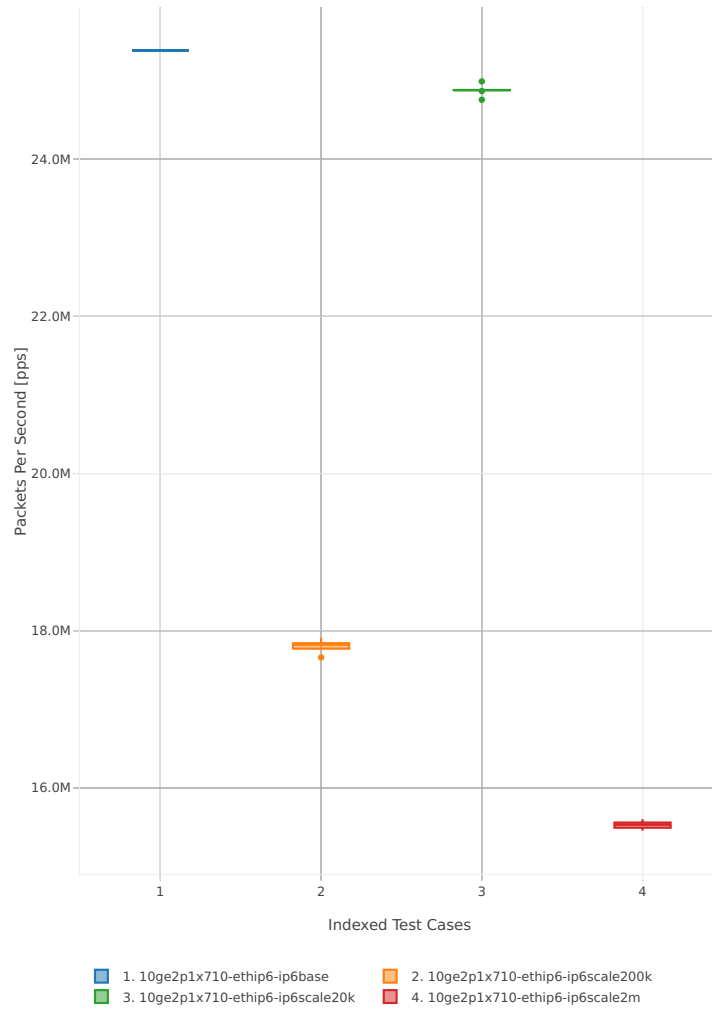


Throughput: ip6-3n-skx-x710-78b-2t1c-base_and_scale-pdr

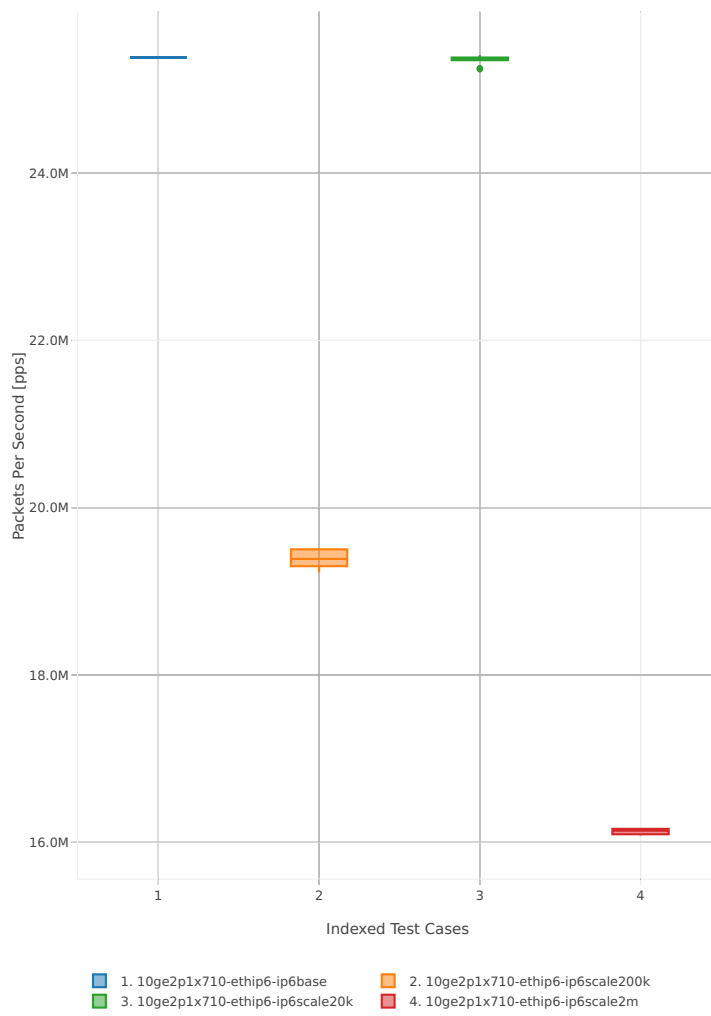


78b-4t2c-base_and_scale

Throughput: ip6-3n-skx-x710-78b-4t2c-base_and_scale-ndr



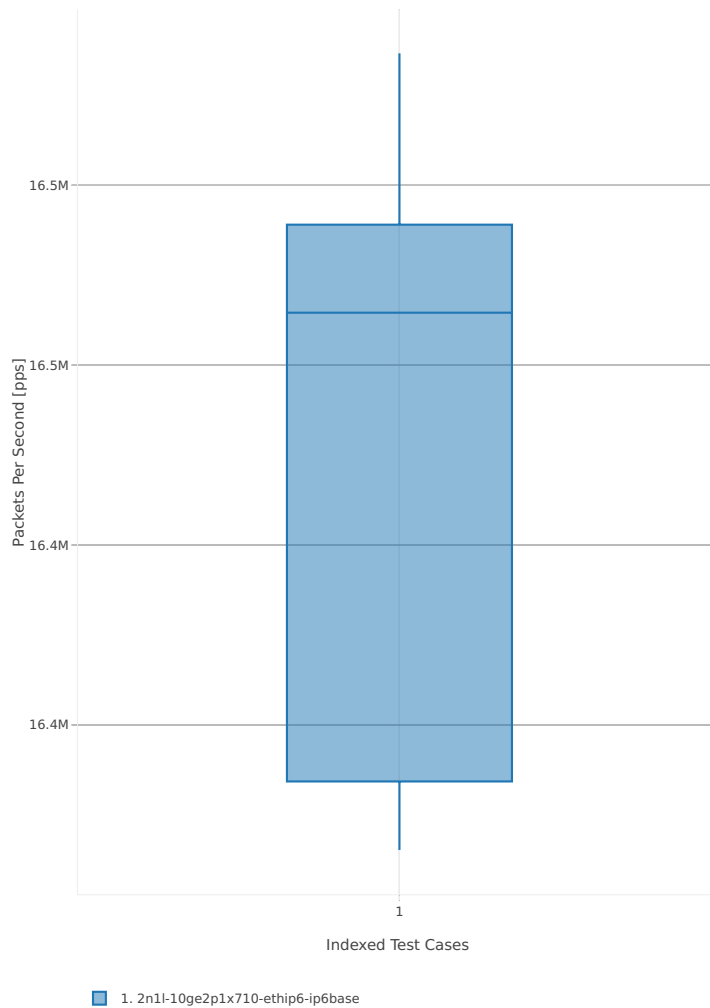
Throughput: ip6-3n-skx-x710-78b-4t2c-base_and_scale-pdr



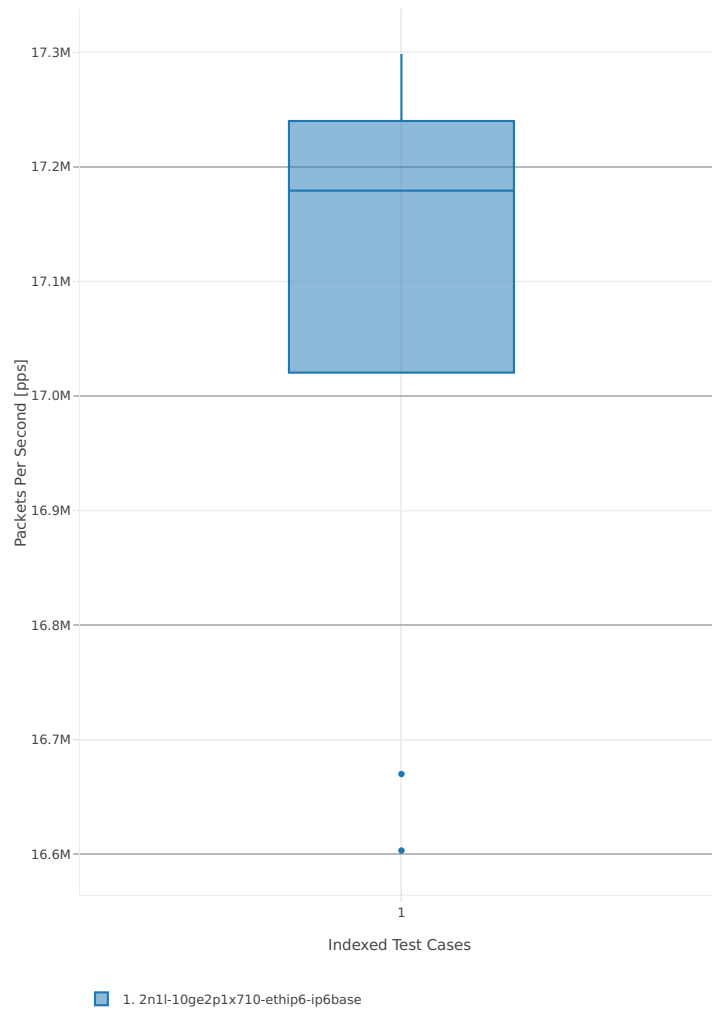
2n-skx-x710

78b-2t1c-base_and_scale

Throughput: ip6-2n-skx-x710-78b-2t1c-base_and_scale-ndr

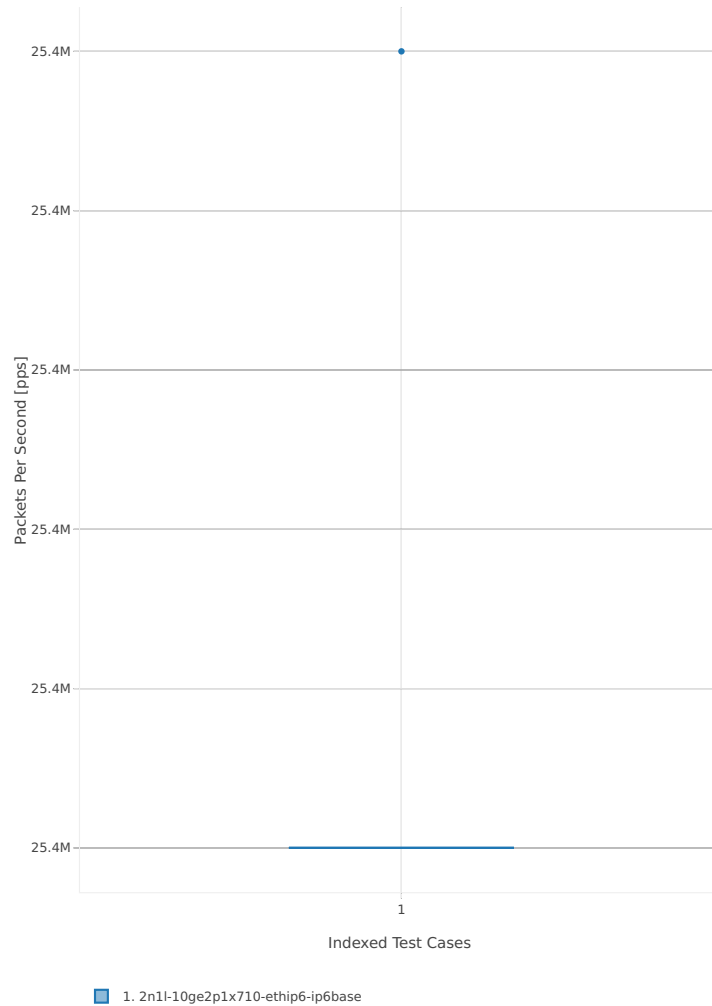


Throughput: ip6-2n-skx-x710-78b-2t1c-base_and_scale-pdr

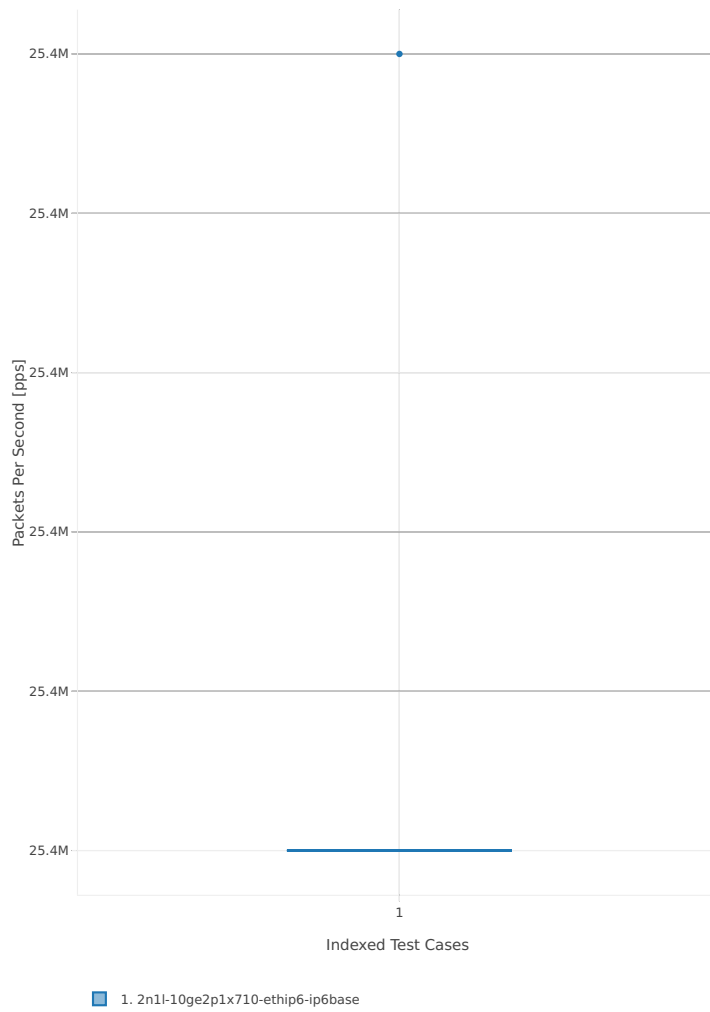


78b-4t2c-base_and_scale

Throughput: ip6-2n-skx-x710-78b-4t2c-base_and_scale-ndr



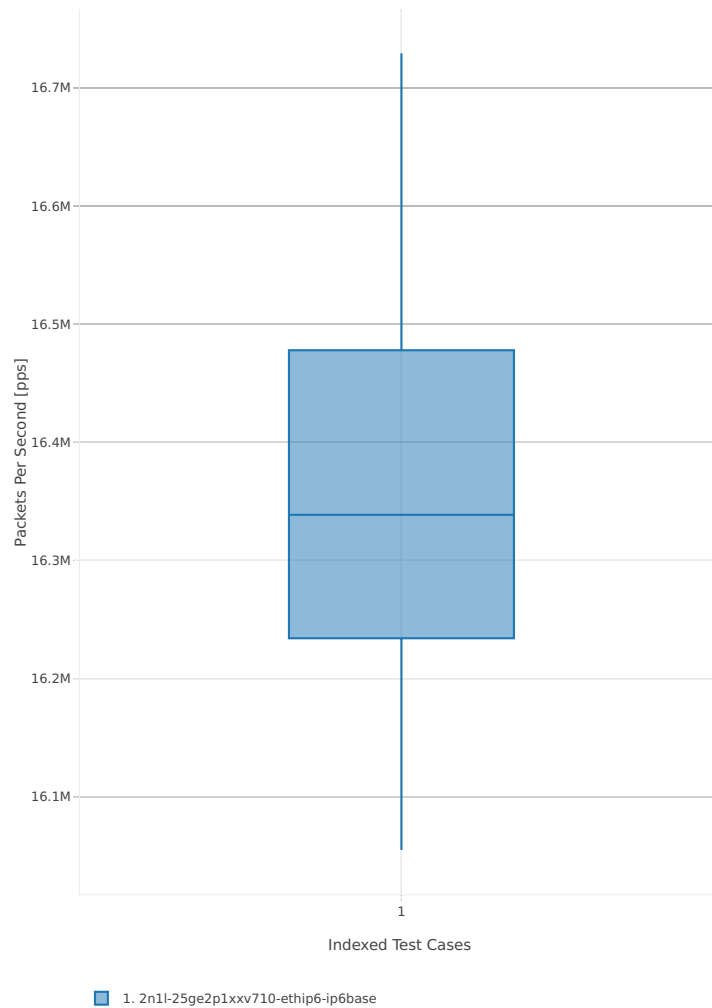
Throughput: ip6-2n-skx-x710-78b-4t2c-base_and_scale-pdr



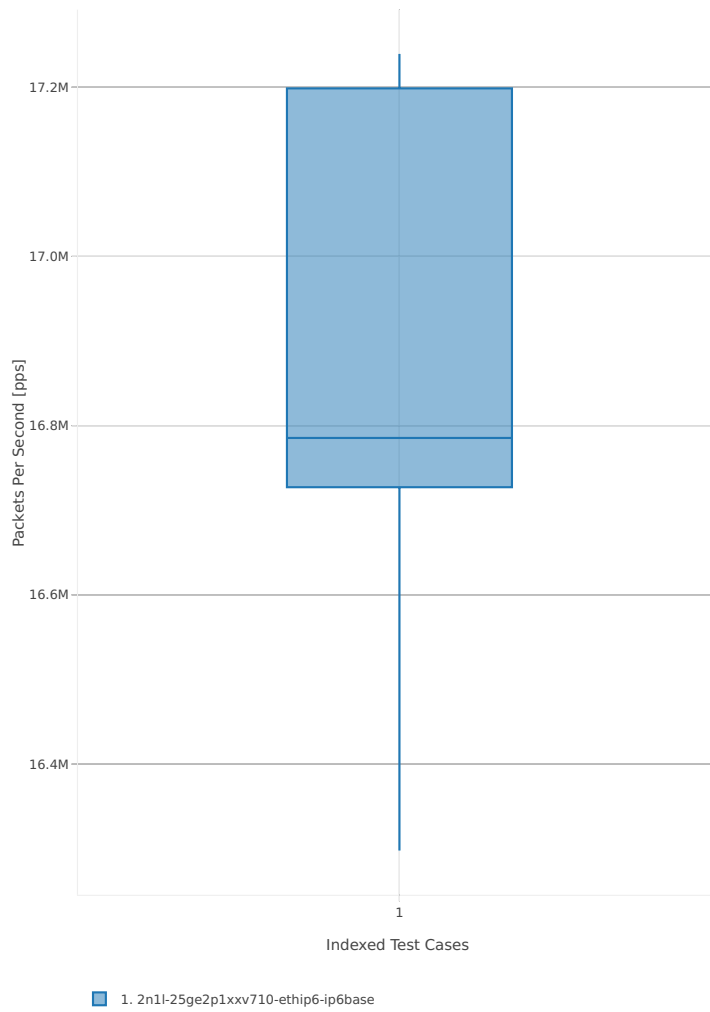
2n-skx-xxv710

78b-2t1c-base_and_scale

Throughput: ip6-2n-skx-xxv710-78b-2t1c-base_and_scale-ndr

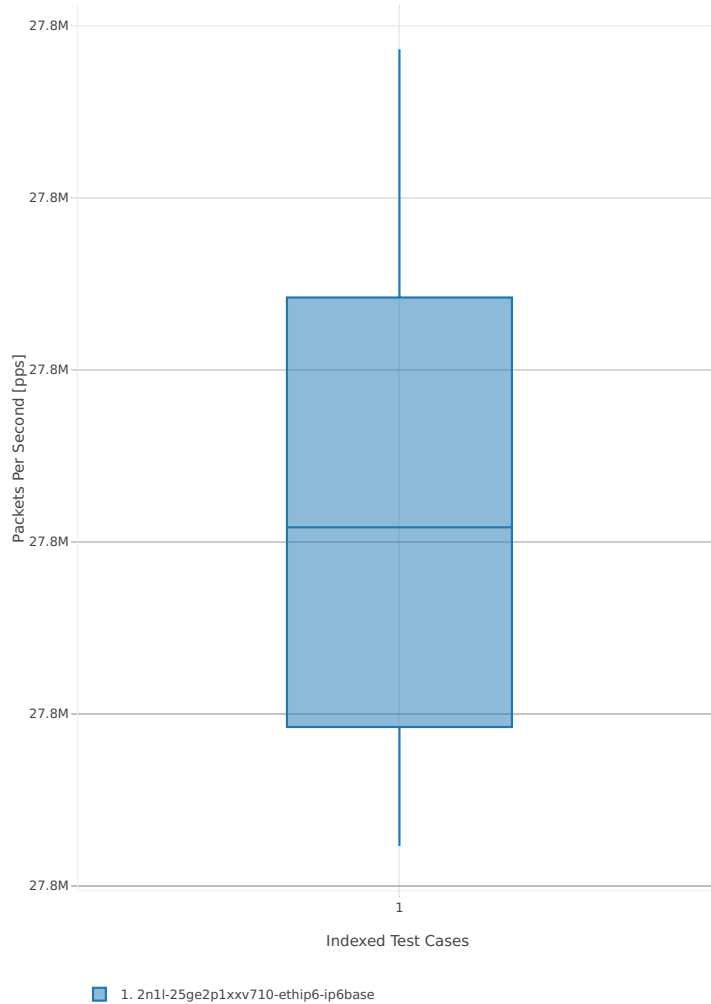


Throughput: ip6-2n-skx-xxv710-78b-2t1c-base_and_scale-pdr

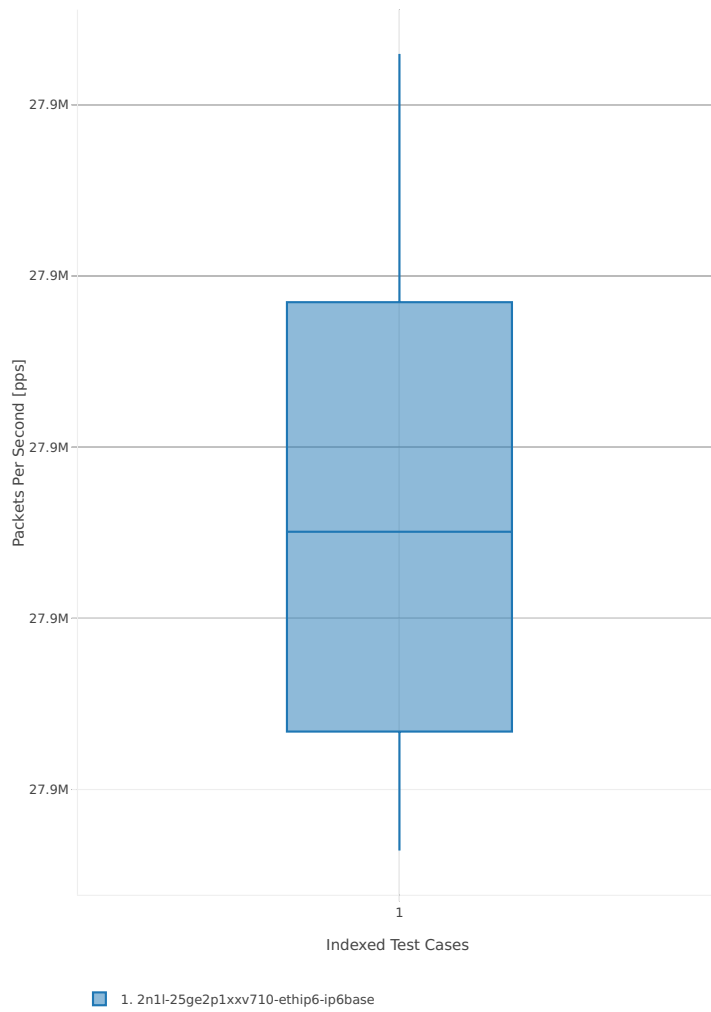


78b-4t2c-base_and_scale

Throughput: ip6-2n-skx-xxv710-78b-4t2c-base_and_scale-ndr



Throughput: ip6-2n-skx-xxv710-78b-4t2c-base_and_scale-pdr



2.3.4 SRv6 Routing

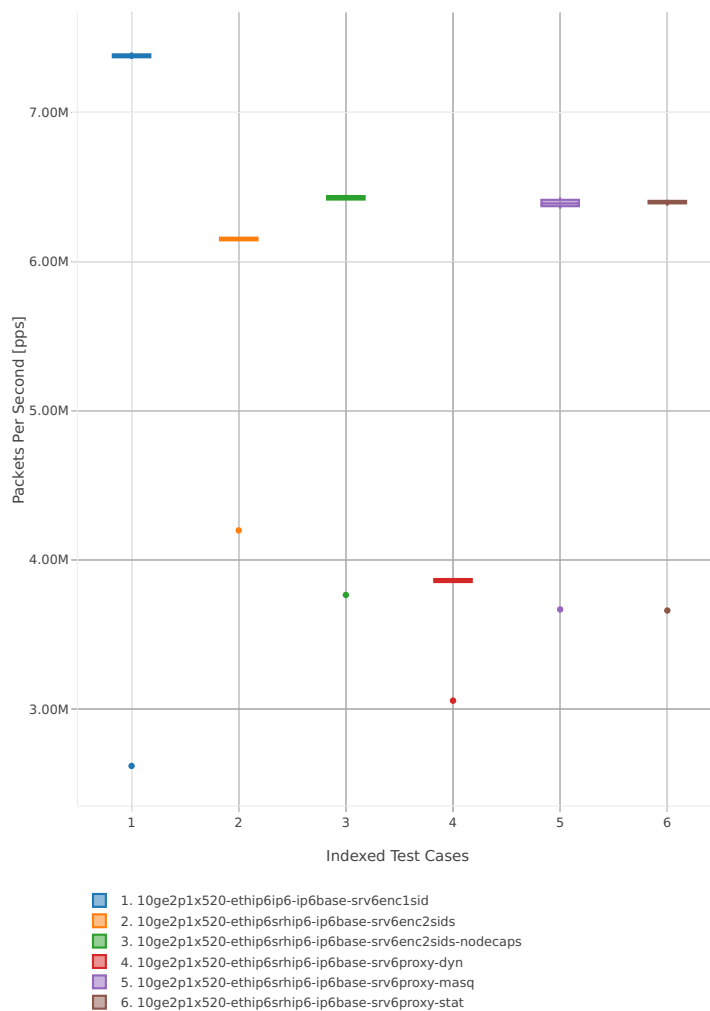
Following sections include summary graphs of VPP Phy-to-Phy performance with SRv6, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)³².

3n-hsw-x520

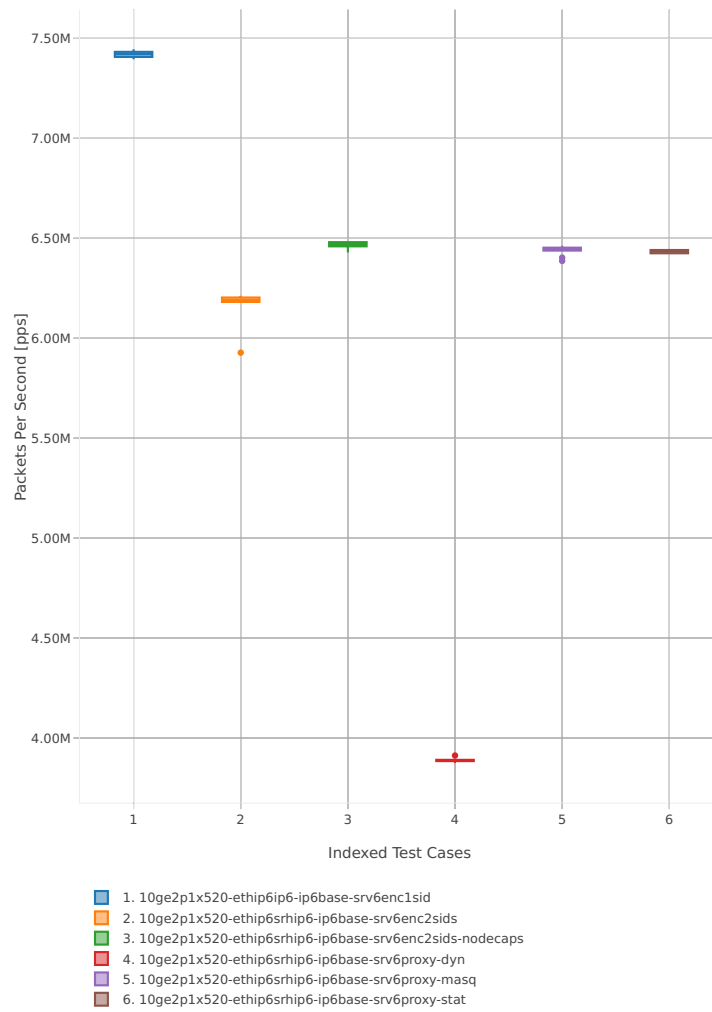
78b-1t1c-features

Throughput: srv6-3n-hsw-x520-78b-1t1c-features-ndr



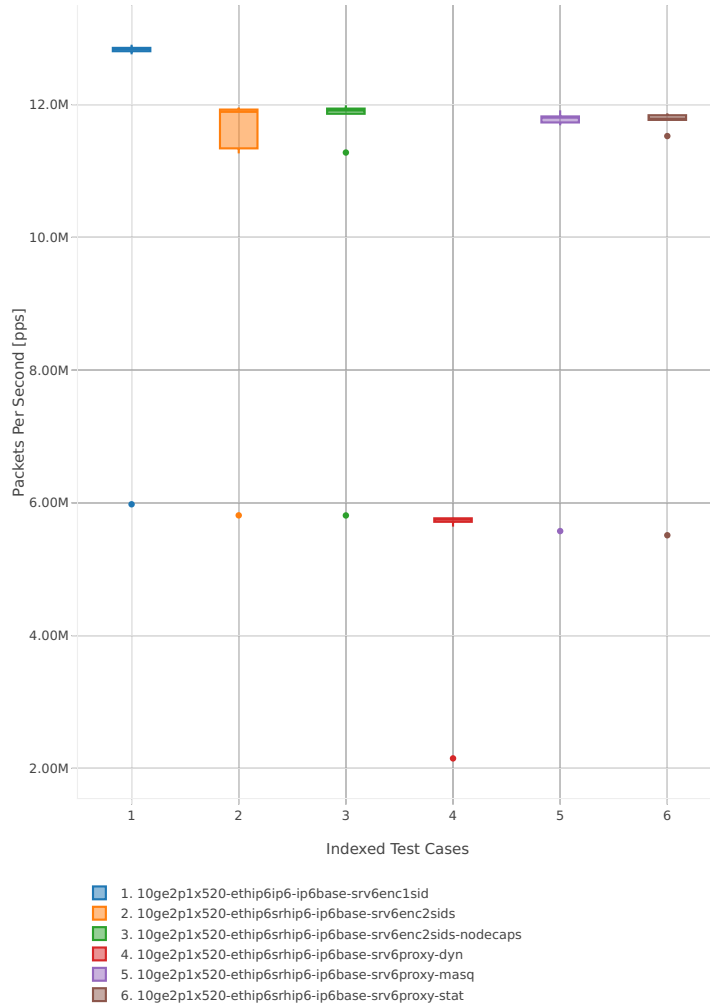
³² <https://git.fd.io/csit/tree/tests/vpp/perf/srv6?h=rls1807>

Throughput: srv6-3n-hsw-x520-78b-1t1c-features-pdr

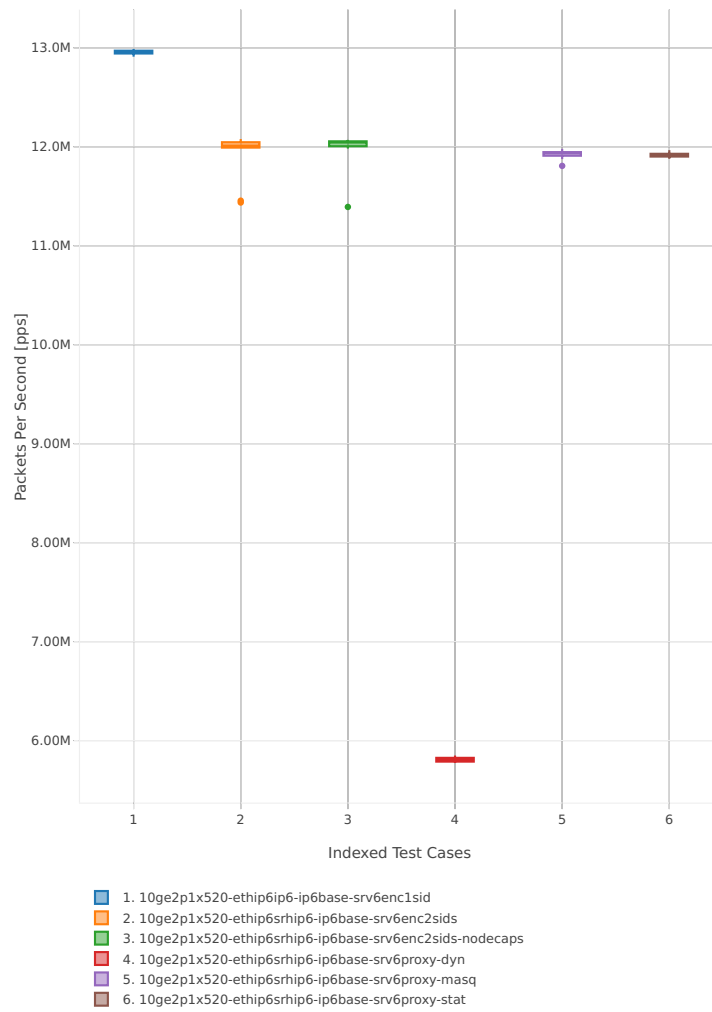


78b-2t2c-features

Throughput: srv6-3n-hsw-x520-78b-2t2c-features-ndr



Throughput: srv6-3n-hsw-x520-78b-2t2c-features-pdr



2.3.5 IPv4 Tunnels

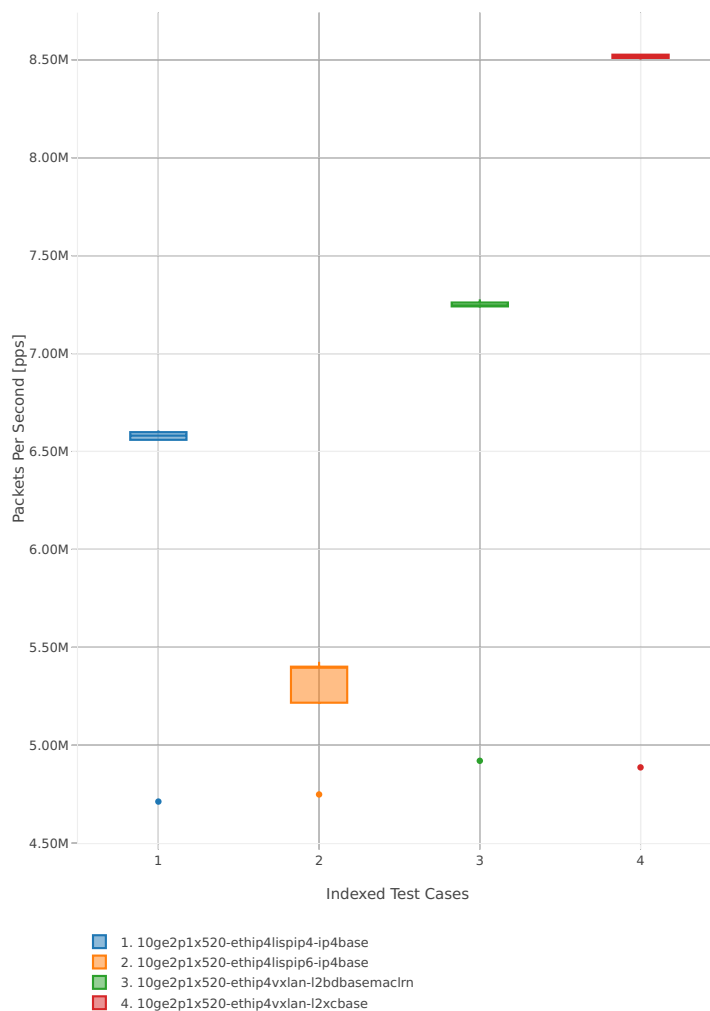
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv4 Overlay Tunnels, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s),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

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)³³.

3n-hsw-x520

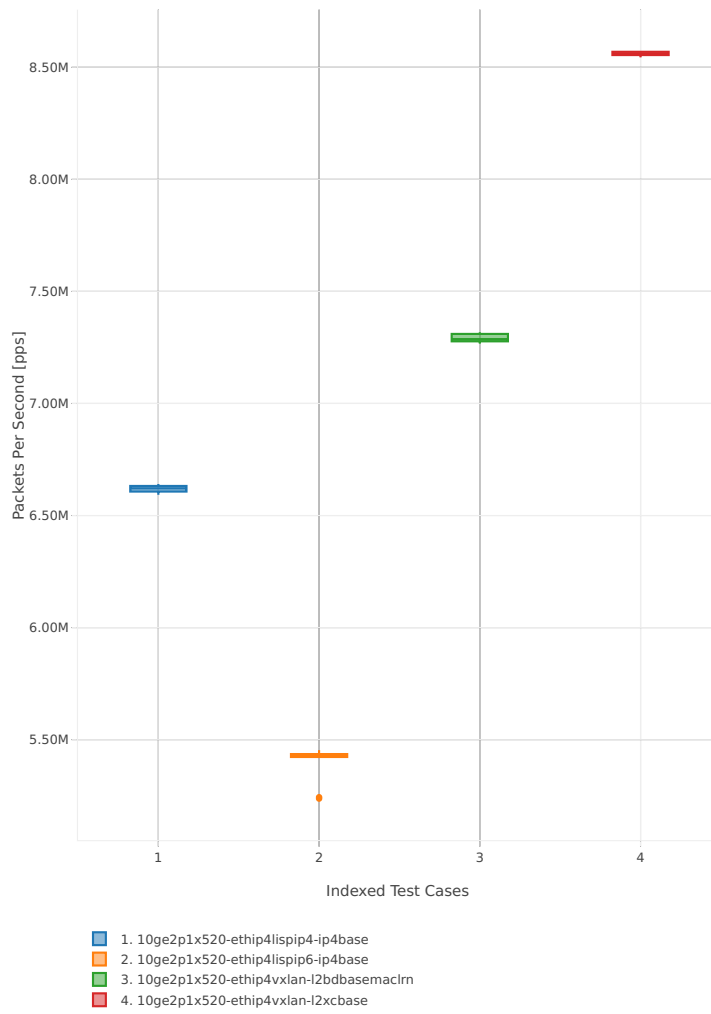
64b-1t1c-base_and_scale

Throughput: ip4tun-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



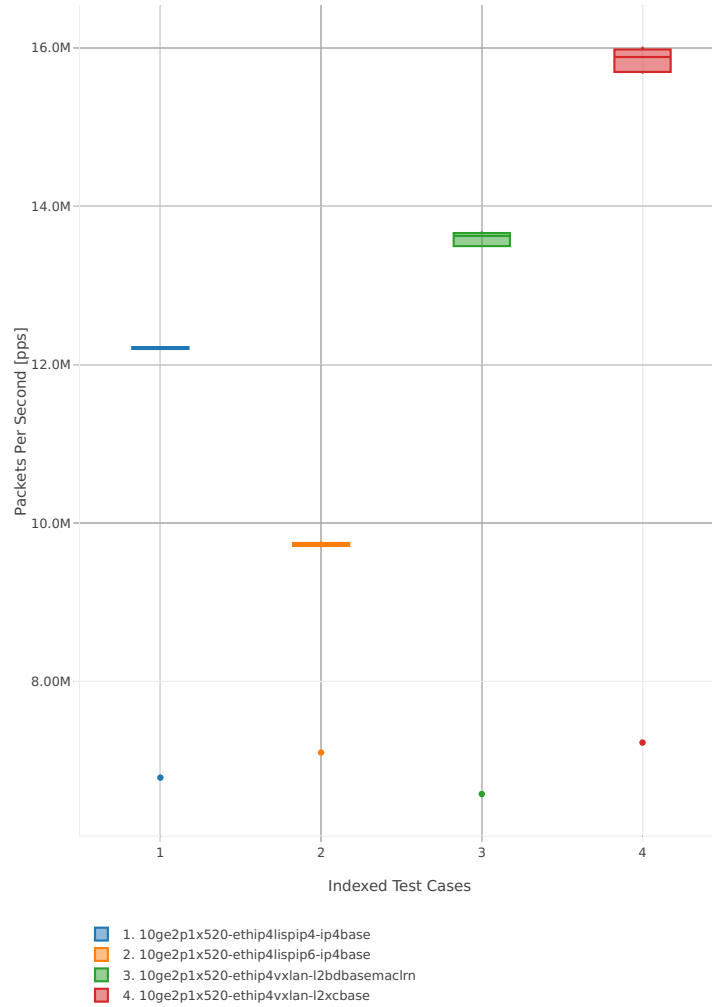
³³ https://git.fd.io/csit/tree/tests/vpp/perf/ip4_tunnels?h=rls1807

Throughput: ip4tun-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

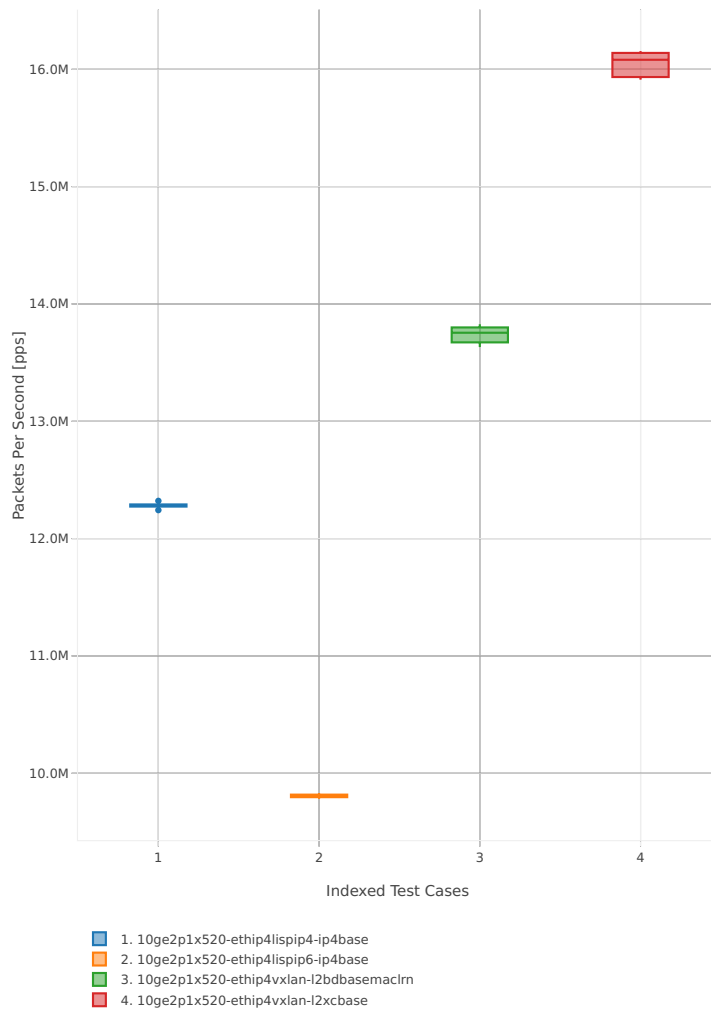


64b-2t2c-base_and_scale

Throughput: ip4tun-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



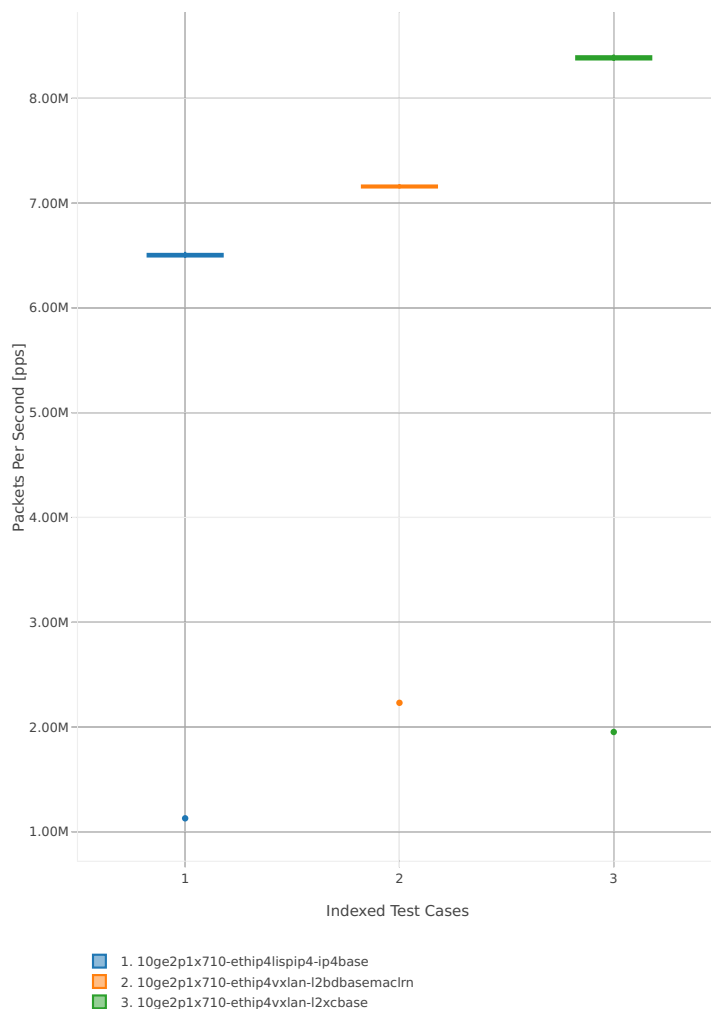
Throughput: ip4tun-3n-hsw-x520-64b-2t2c-base_and_scale-pdr



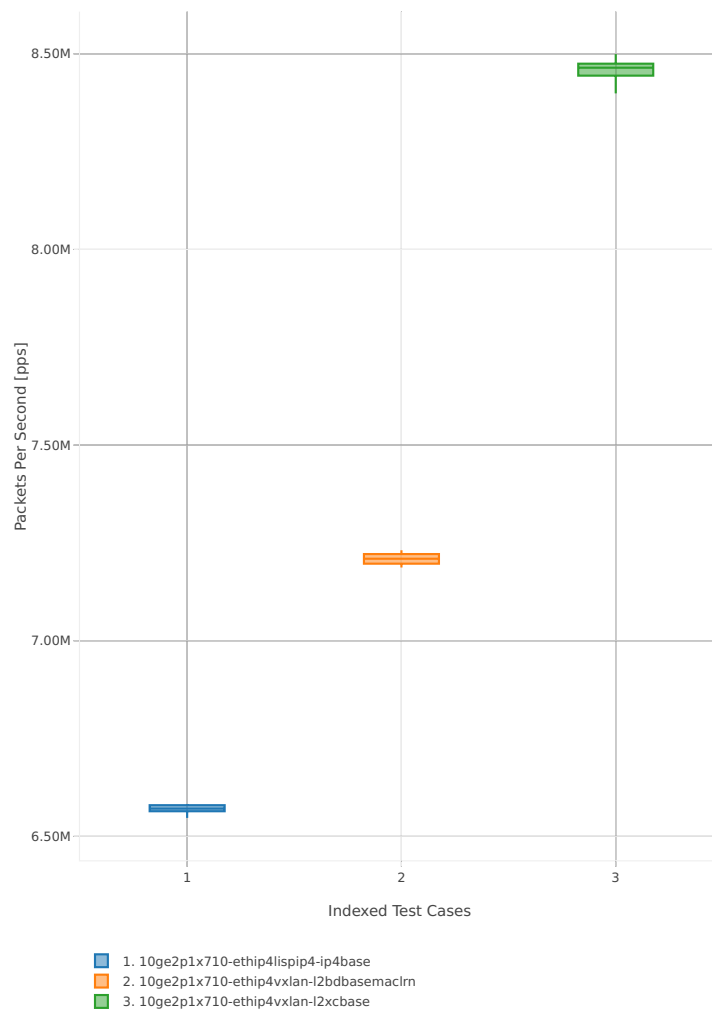
3n-hsw-x710

64b-1t1c-base_and_scale

Throughput: ip4tun-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

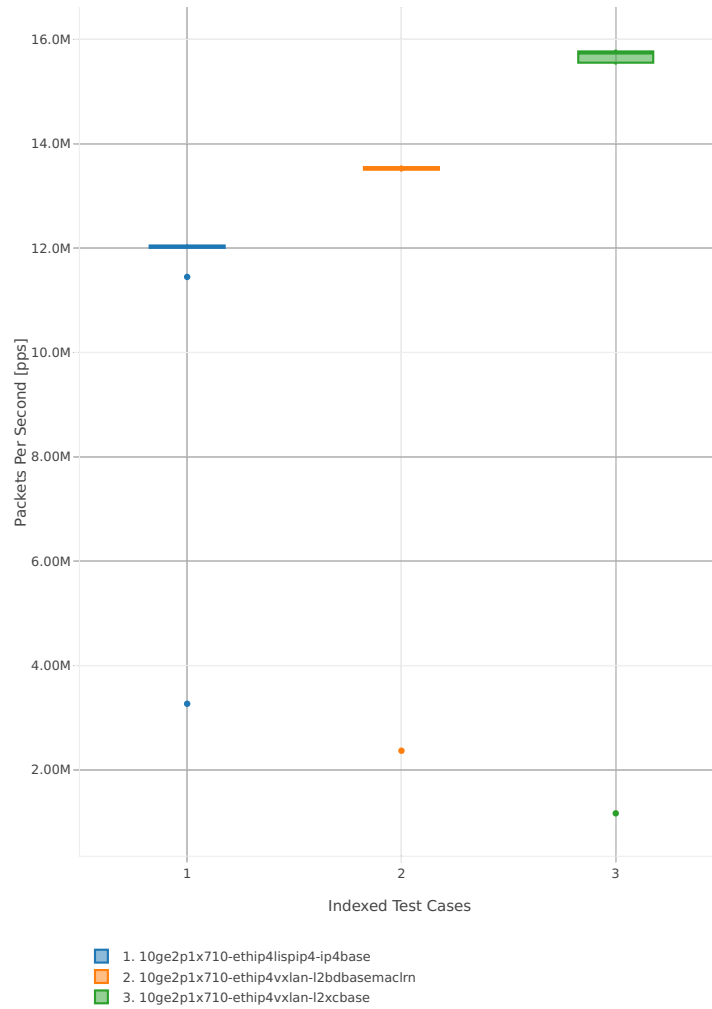


Throughput: ip4tun-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

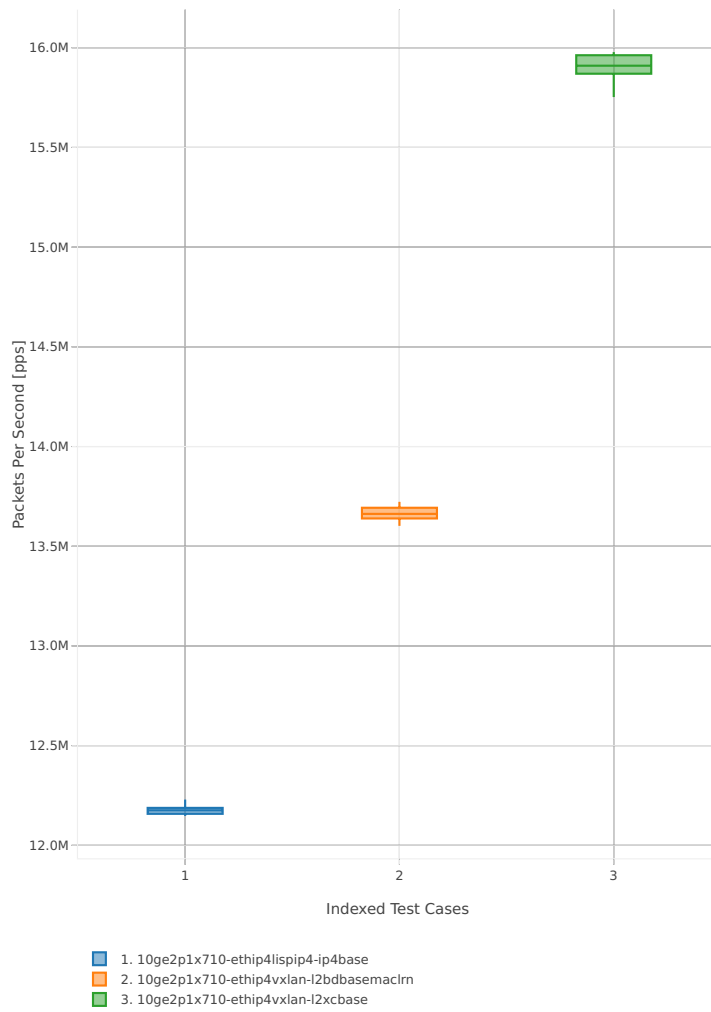


64b-2t2c-base_and_scale

Throughput: ip4tun-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



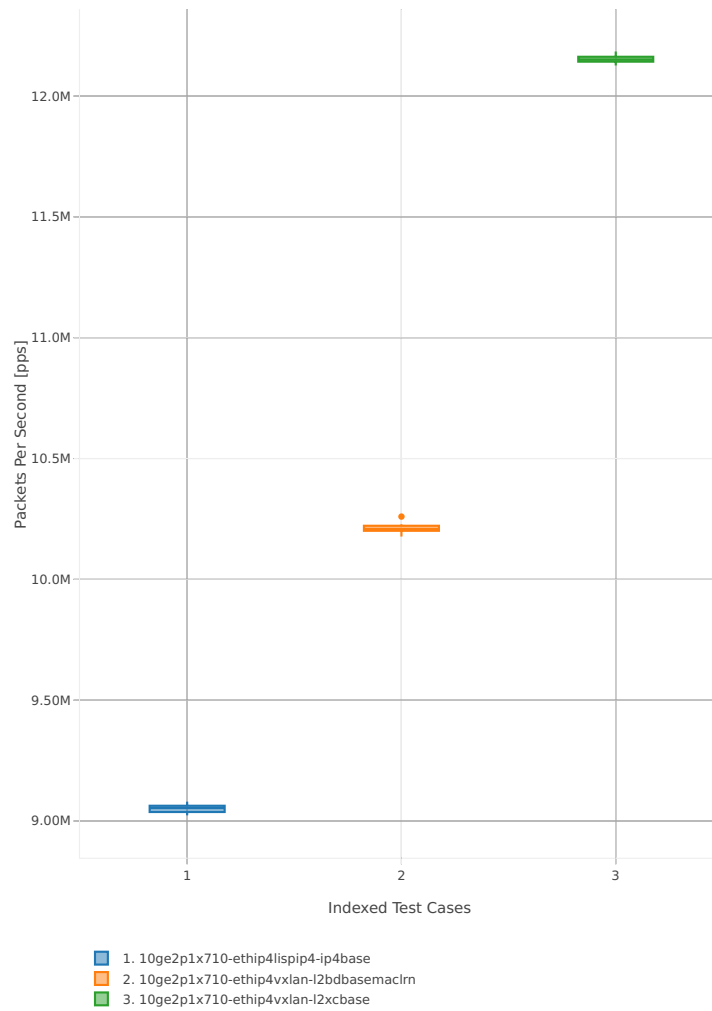
Throughput: ip4tun-3n-hsw-x710-64b-2t2c-base_and_scale-pdr



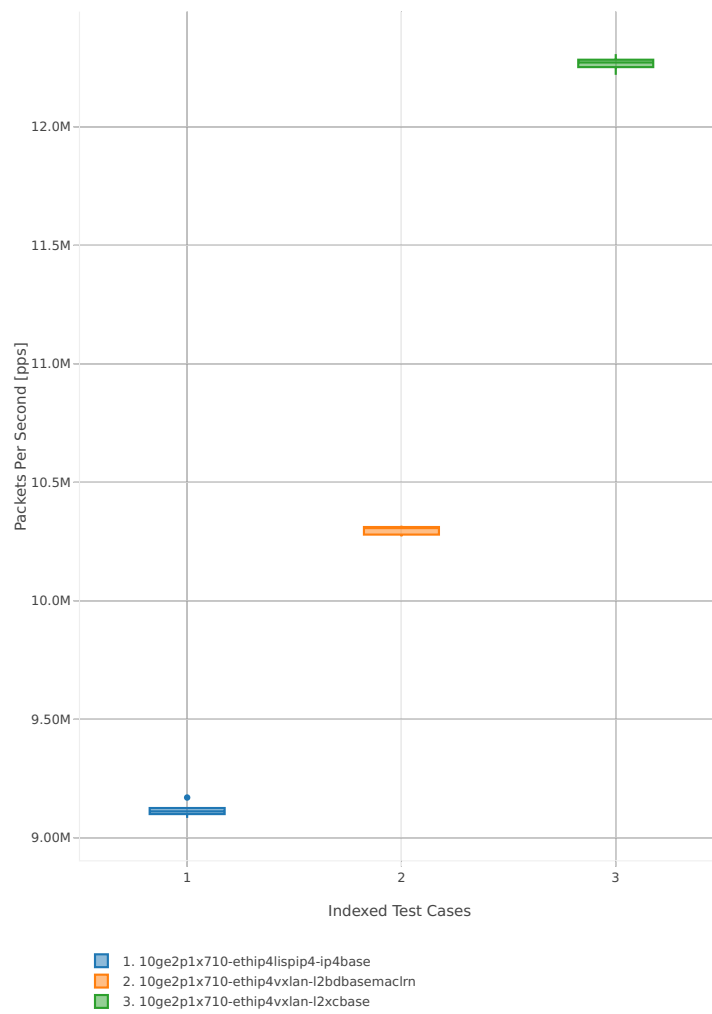
3n-skx-x710

64b-2t1c-base_and_scale

Throughput: ip4tun-3n-skx-x710-64b-2t1c-base_and_scale-ndr

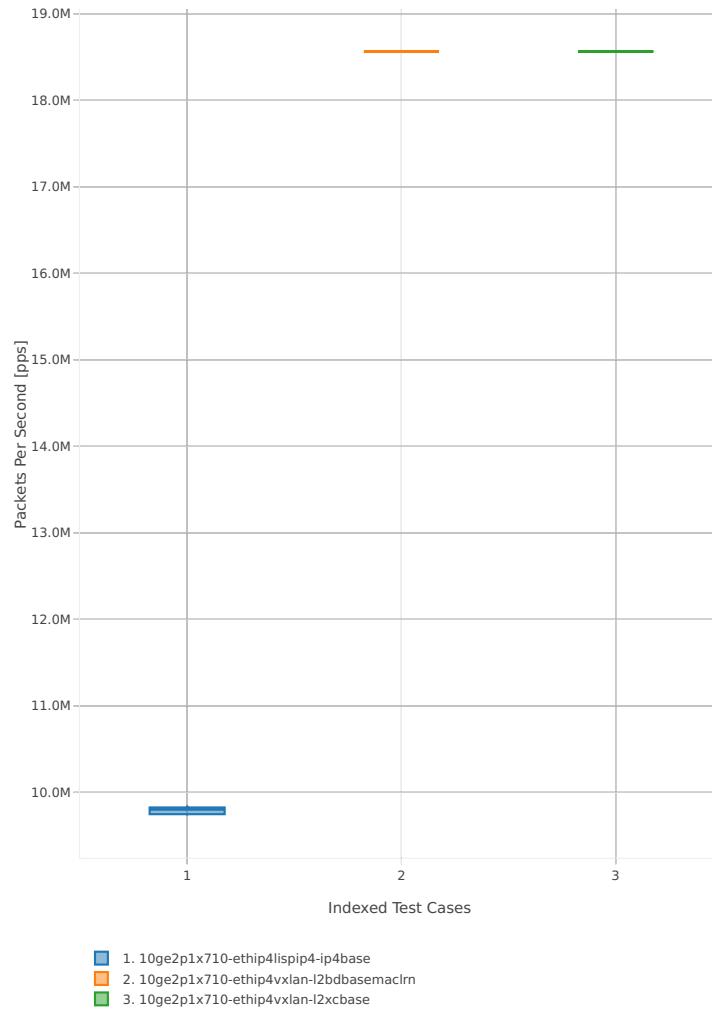


Throughput: ip4tun-3n-skx-x710-64b-2t1c-base_and_scale-pdr

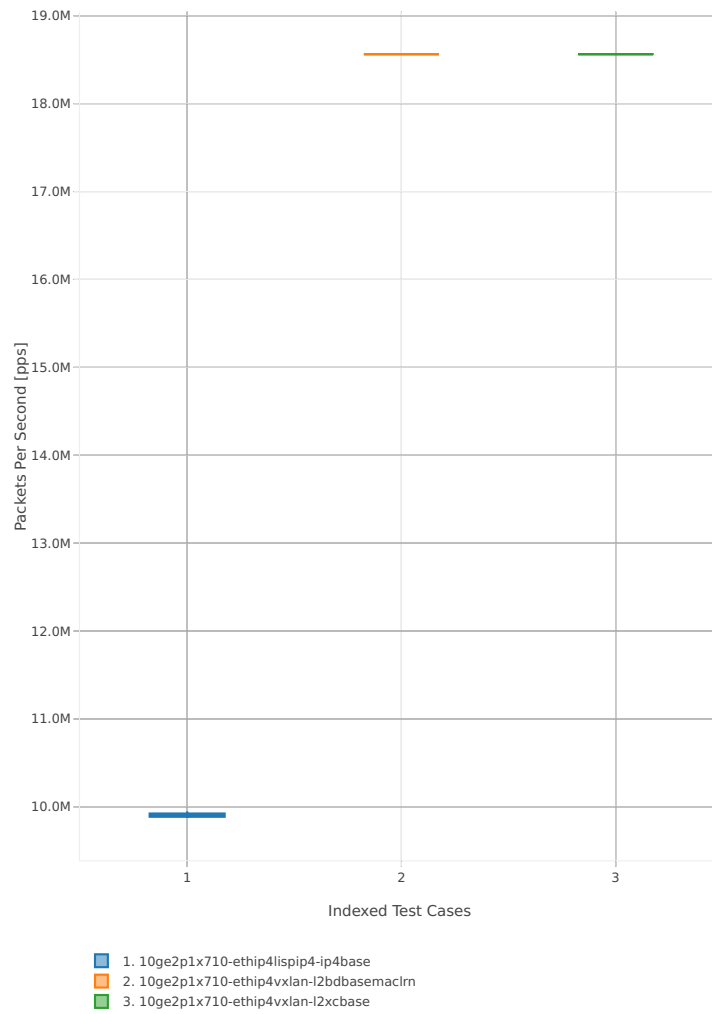


64b-4t2c-base_and_scale

Throughput: ip4tun-3n-skx-x710-64b-4t2c-base_and_scale-ndr



Throughput: ip4tun-3n-skx-x710-64b-4t2c-base_and_scale-pdr



2.3.6 IPv6 Tunnels

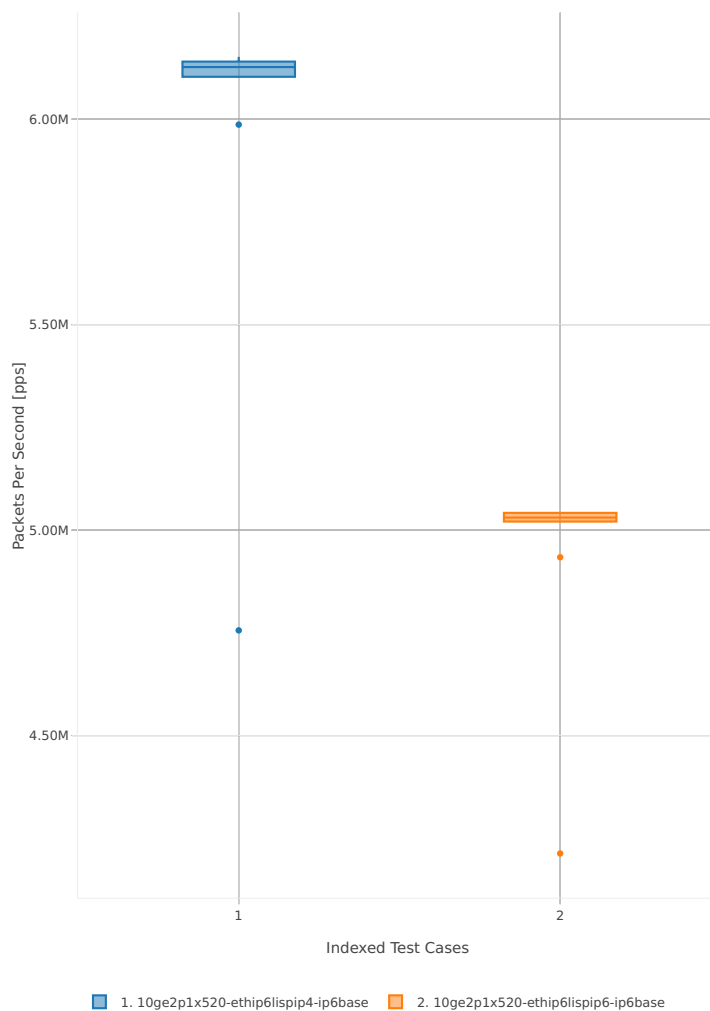
Following sections include summary graphs of VPP Phy-to-Phy performance with IPv6 Overlay Tunnels, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)³⁴.

3n-hsw-x520

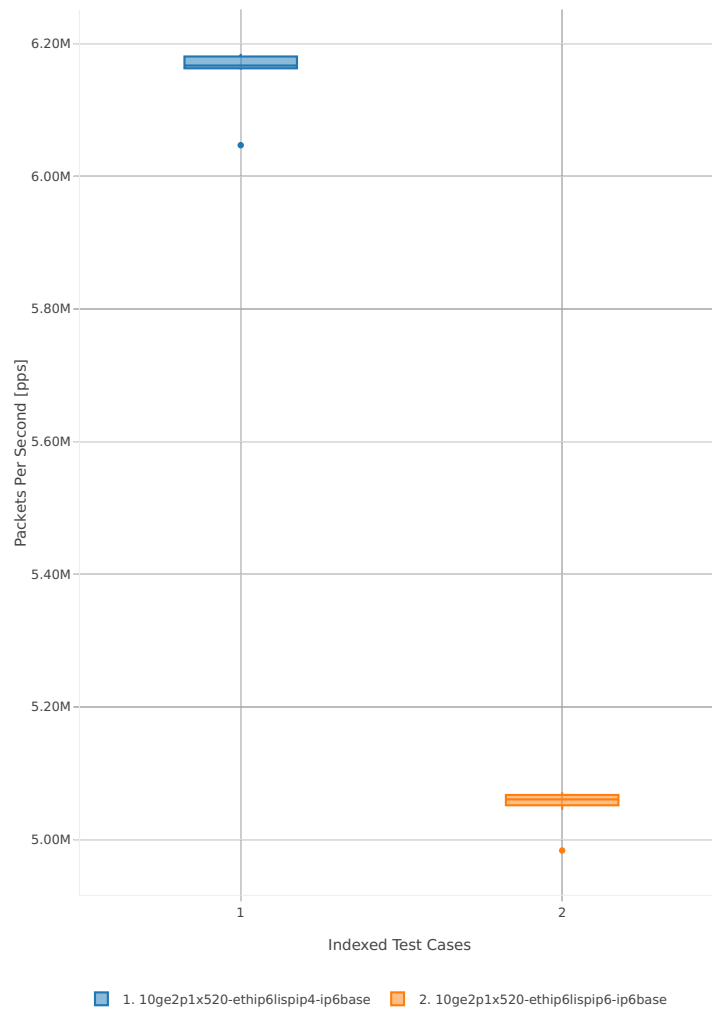
78b-1t1c-base_and_scale

Throughput: ip6tun-3n-hsw-x520-78b-1t1c-base_and_scale-ndr



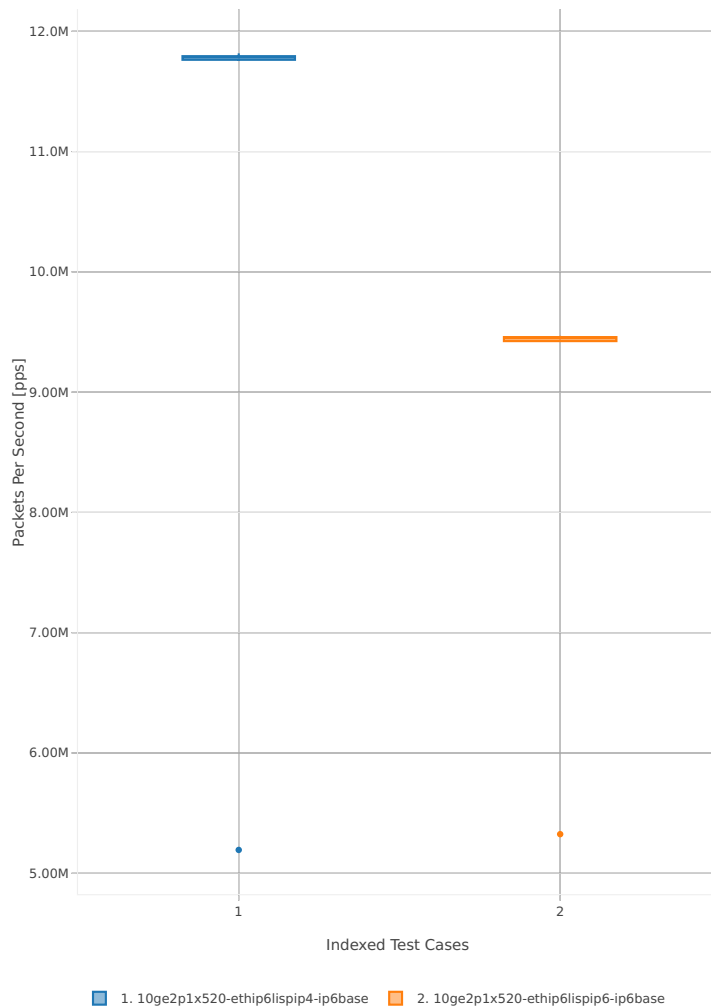
³⁴ https://git.fd.io/csit/tree/tests/vpp/perf/ip6_tunnels?h=rls1807

Throughput: ip6tun-3n-hsw-x520-78b-1t1c-base_and_scale-pdr

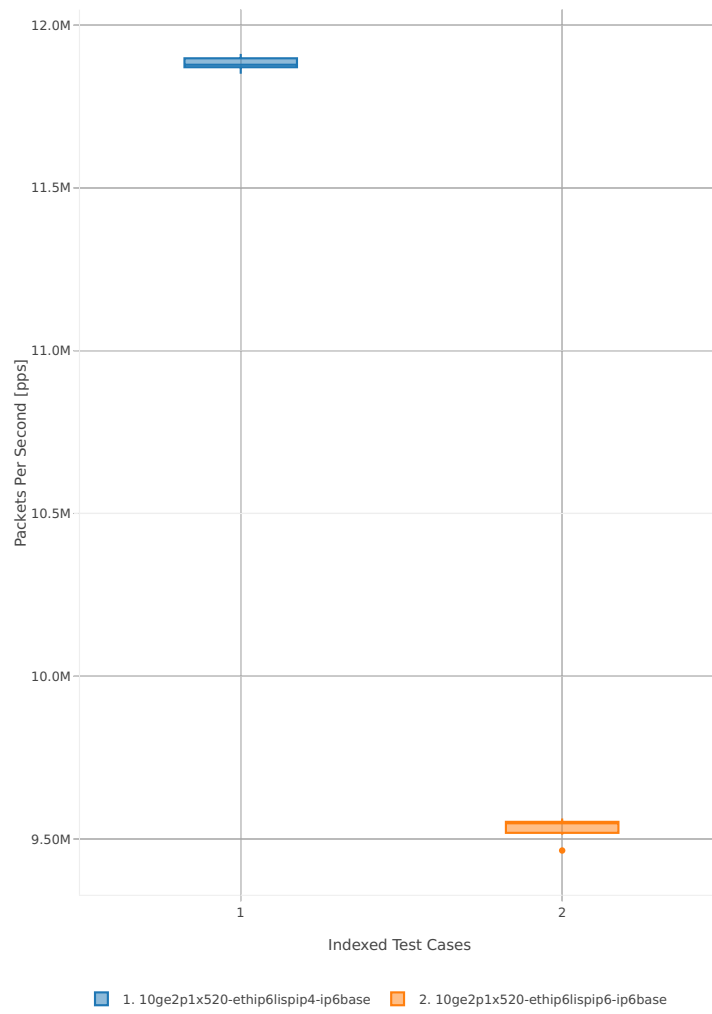


78b-2t2c-base_and_scale

Throughput: ip6tun-3n-hsw-x520-78b-2t2c-base_and_scale-ndr



Throughput: ip6tun-3n-hsw-x520-78b-2t2c-base_and_scale-pdr



2.3.7 KVM VMs vhost-user

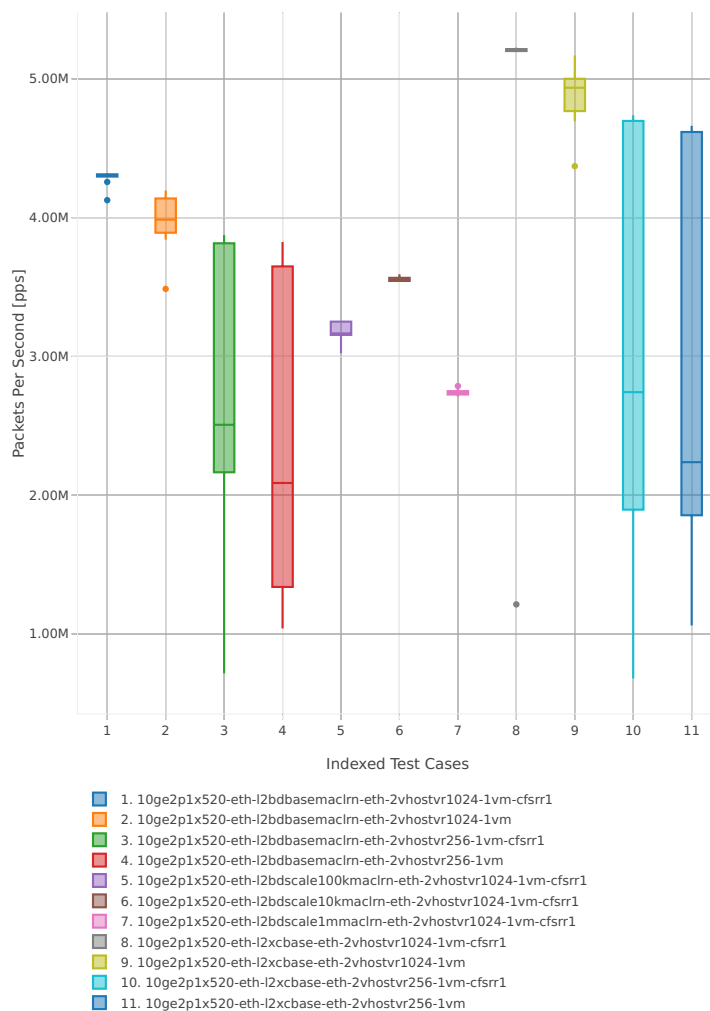
Following sections include summary graphs of VPP Phy-to-VM(s)-to-Phy performance with VM virtio and VPP vhost-user virtual interfaces, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), 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](#)³⁵.

3n-hsw-x520

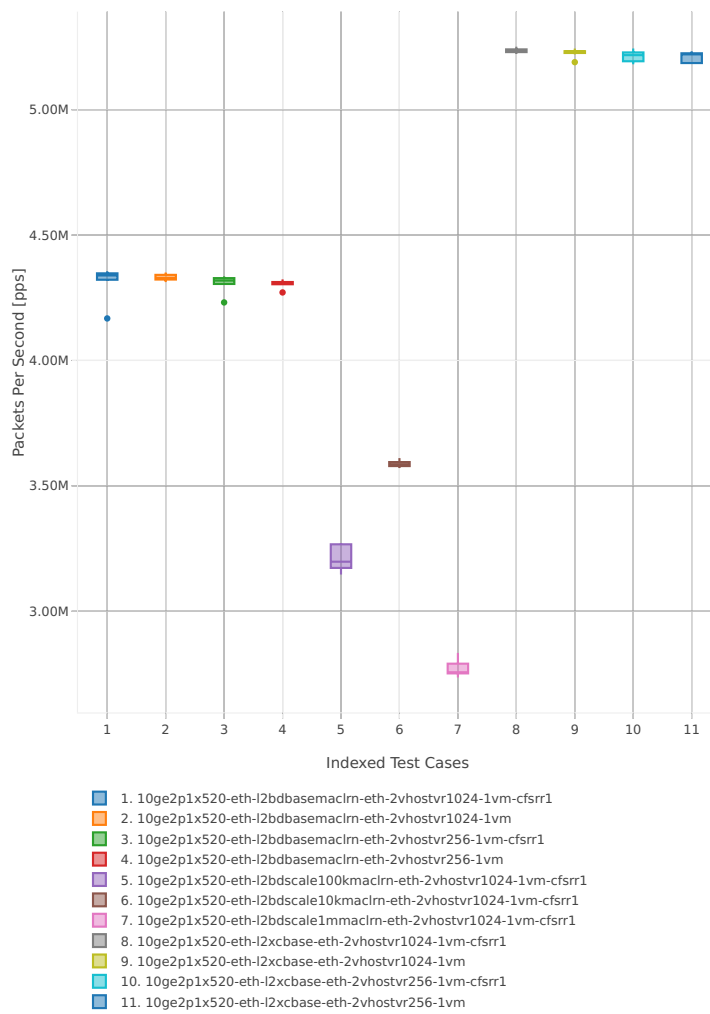
64b-1t1c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



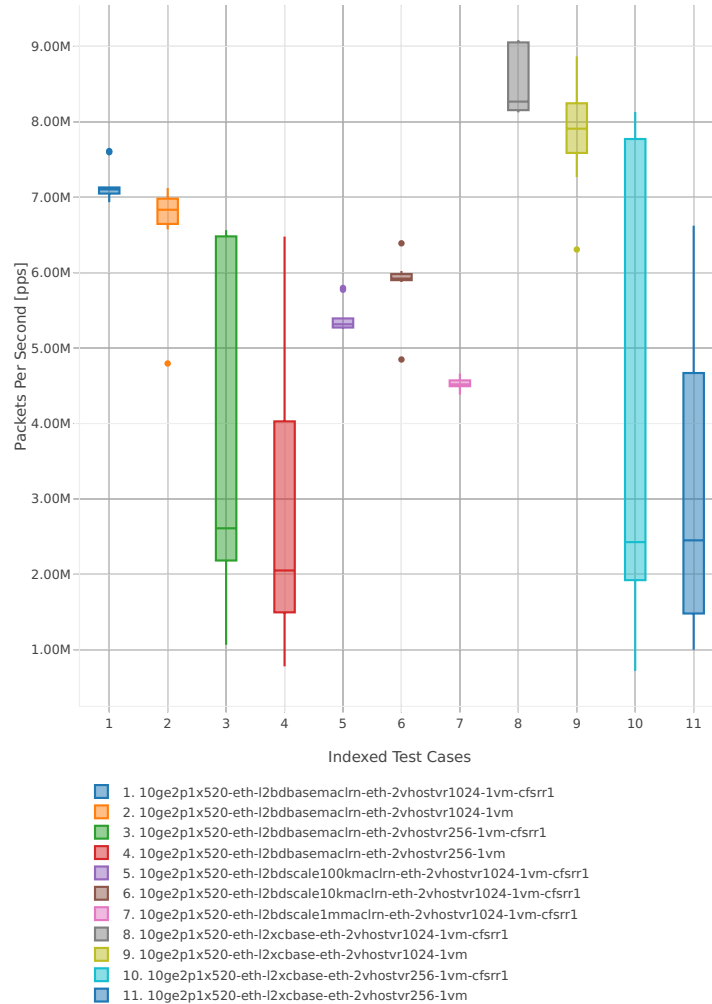
³⁵ https://git.fd.io/csit/tree/tests/vpp/perf/vm_vhost?h=rls1807

Throughput: vhost-l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

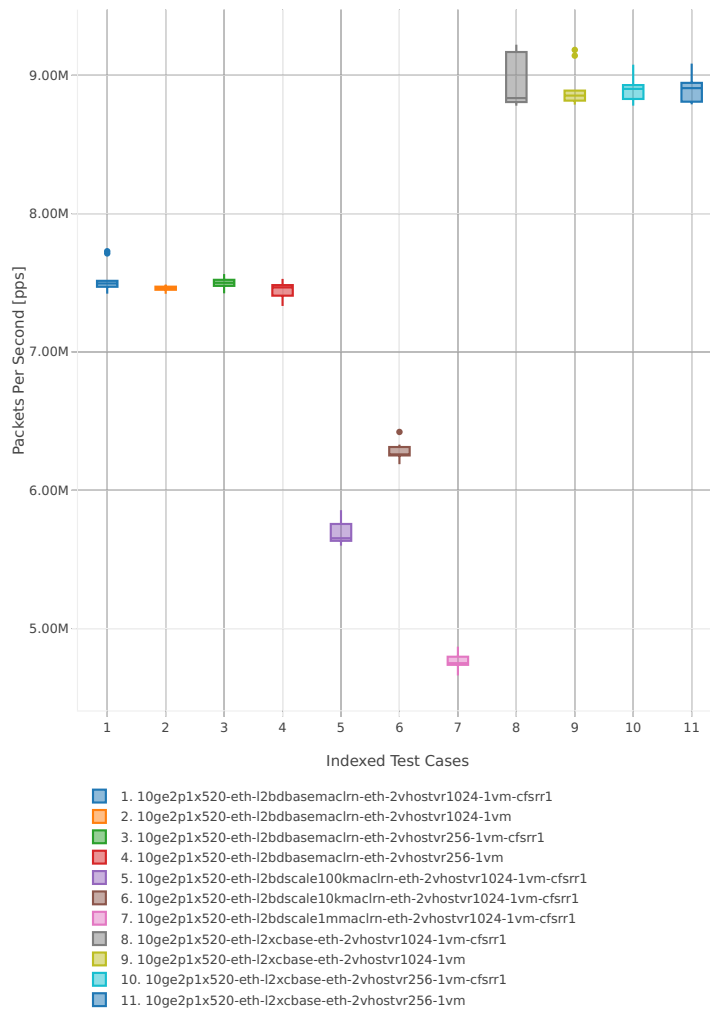


64b-2t2c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-ndr

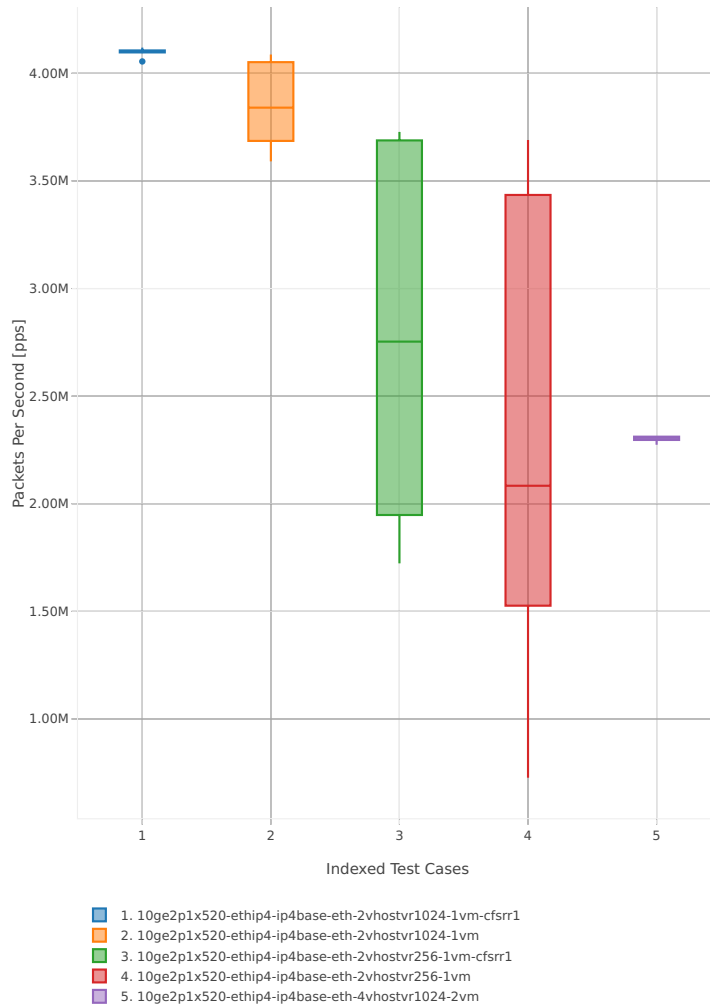


Throughput: vhost-l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-pdr

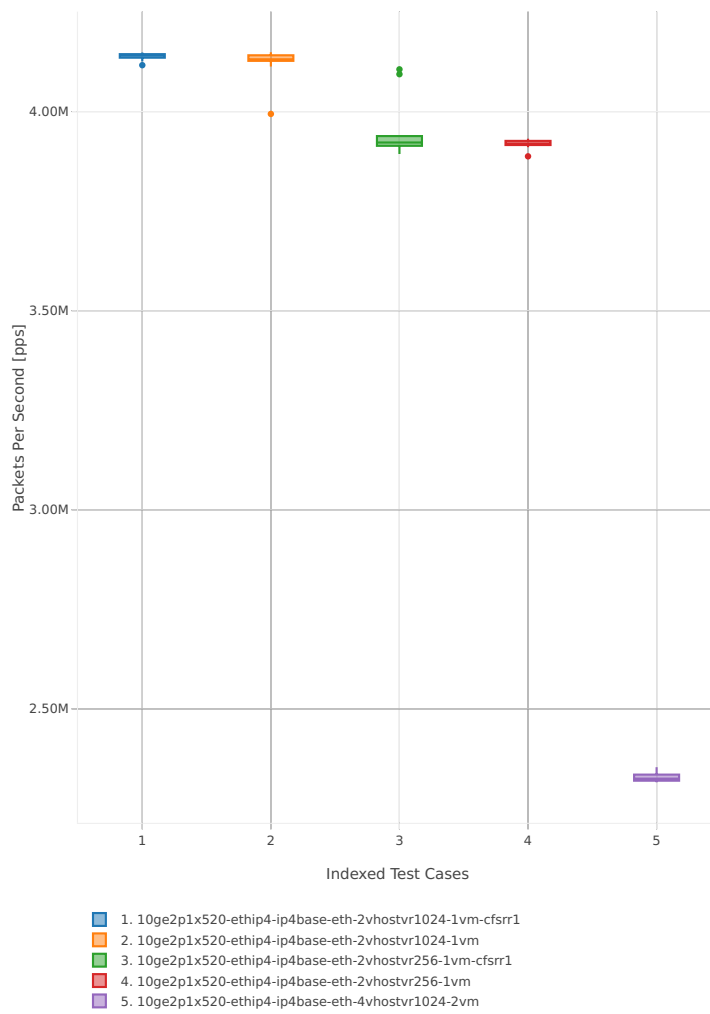


64b-1t1c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-x520-64b-1t1c-base_and_scale-ndr

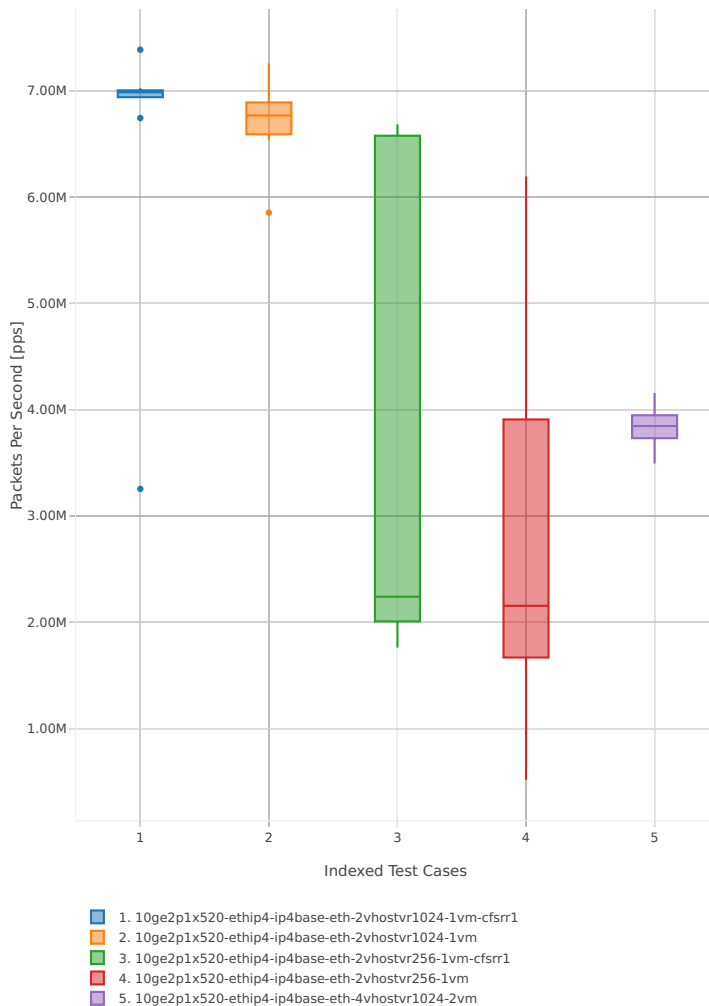


Throughput: vhost-ip4-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

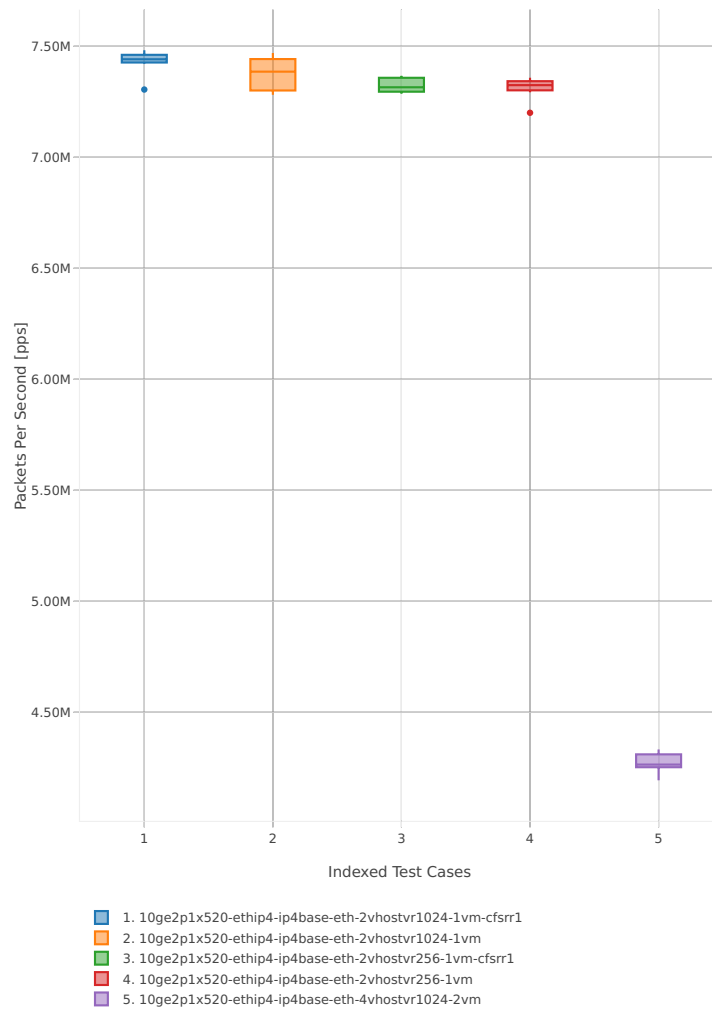


64b-2t2c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



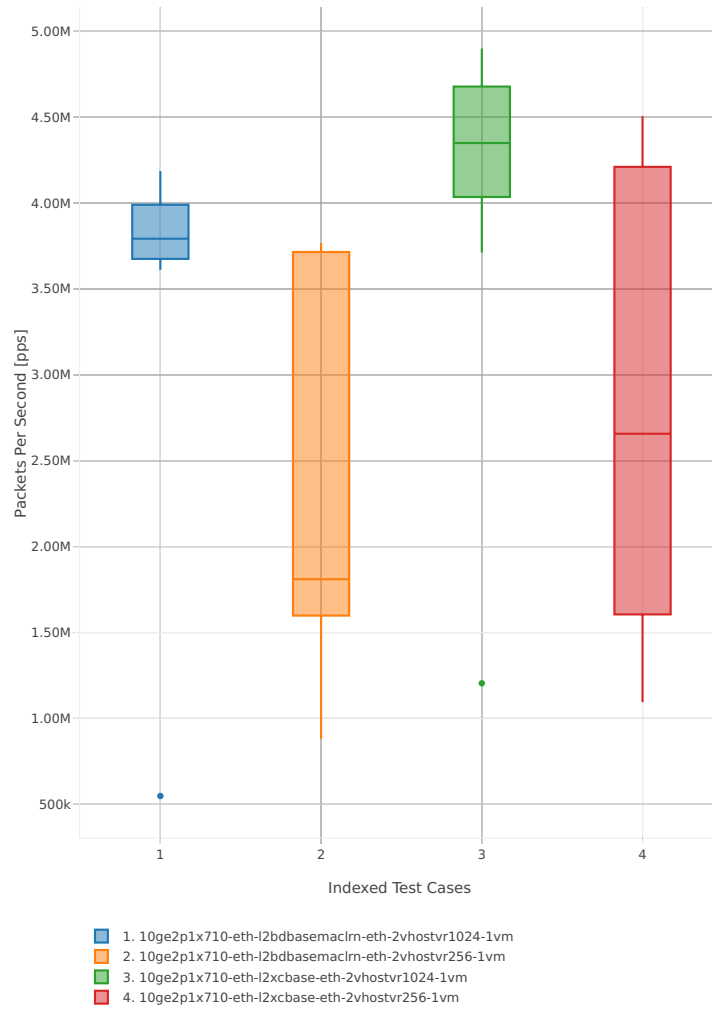
Throughput: vhost-ip4-3n-hsw-x520-64b-2t2c-base_and_scale-pdr



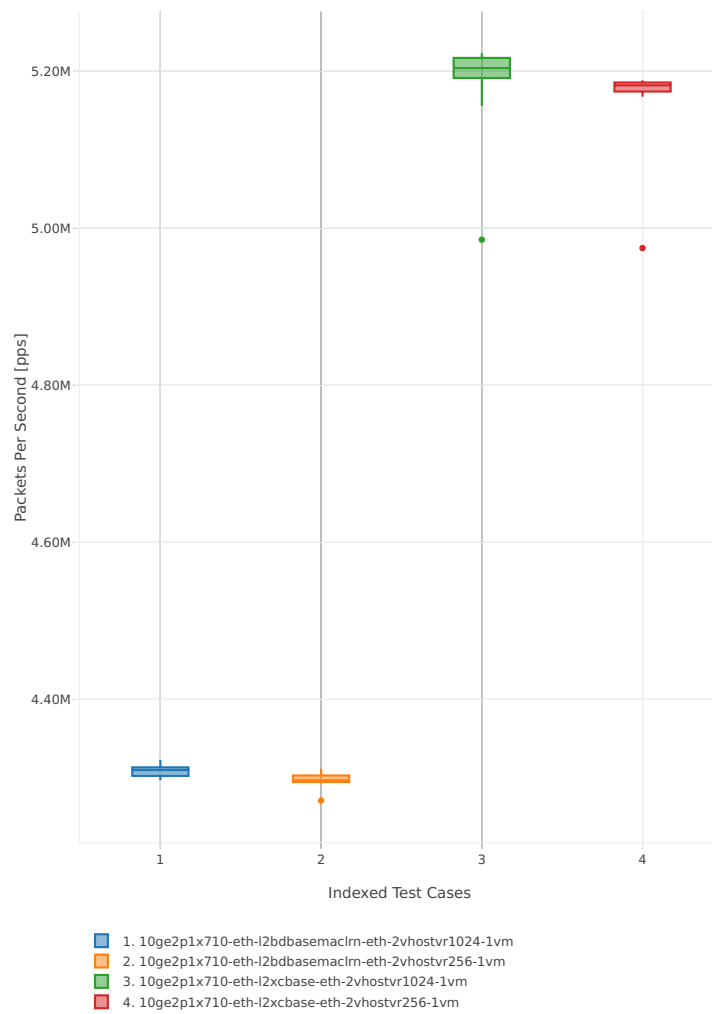
3n-hsw-x710

64b-1t1c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

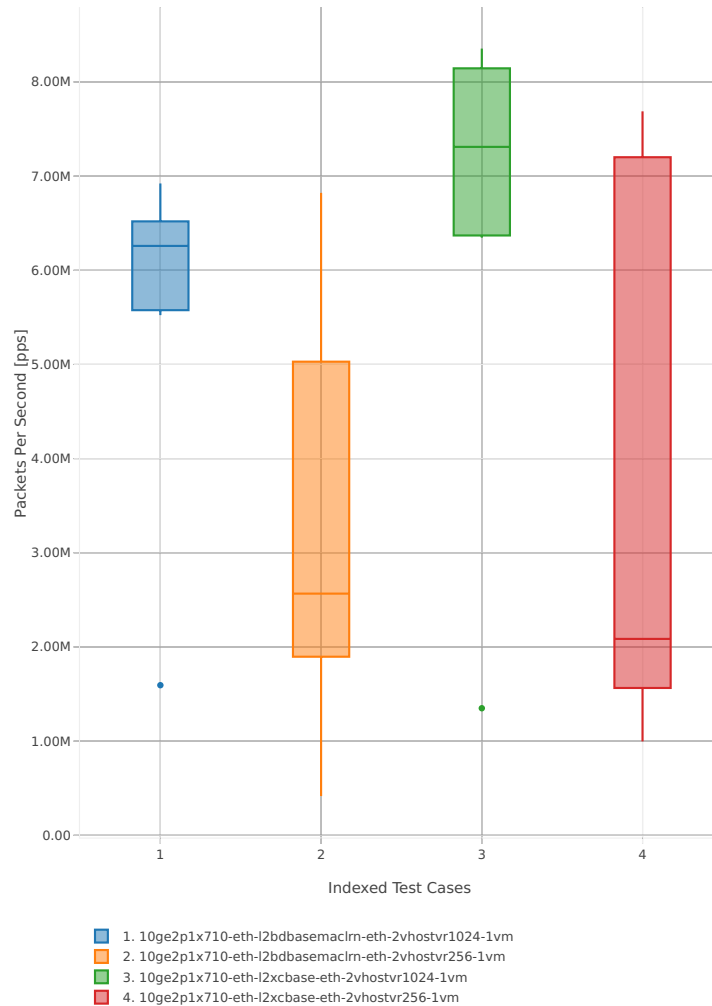


Throughput: vhost-l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

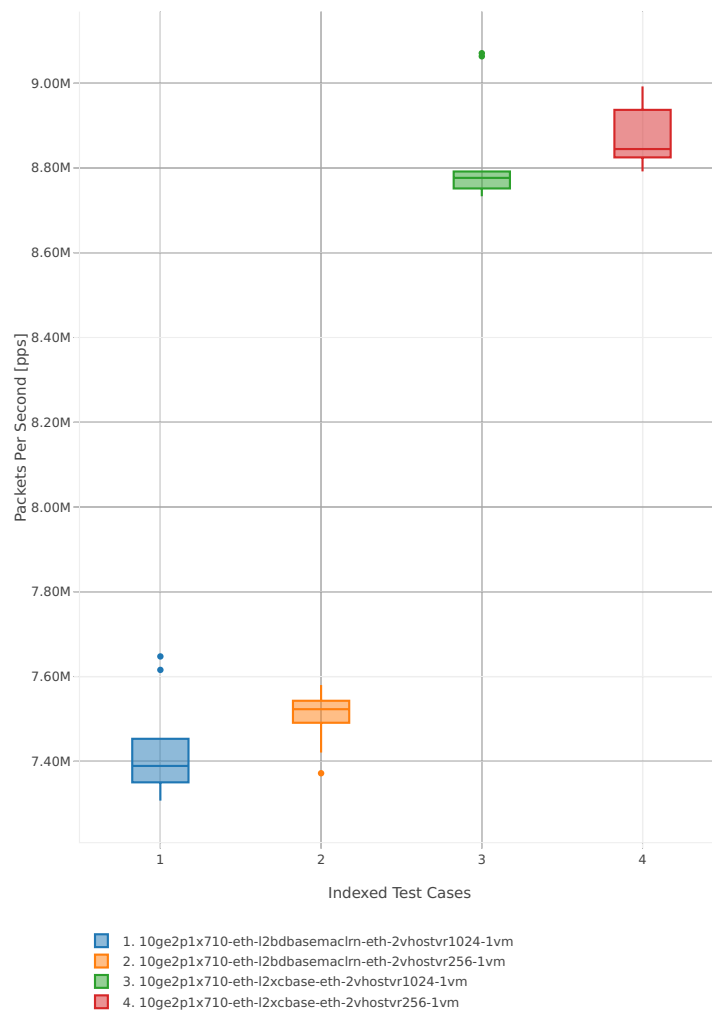


64b-2t2c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-ndr

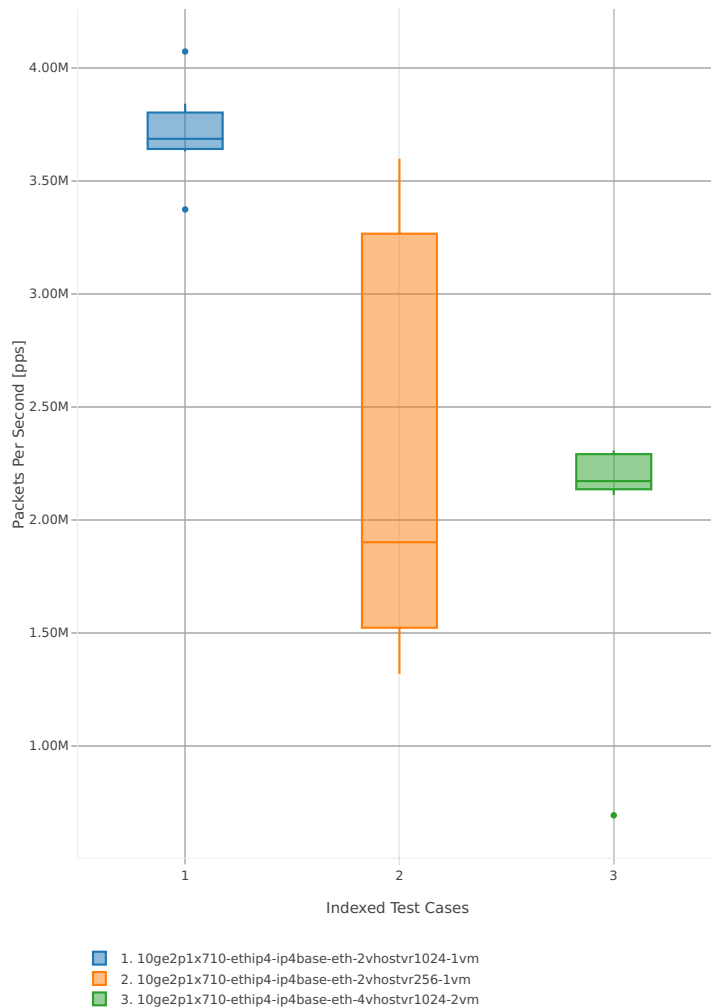


Throughput: vhost-l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-pdr

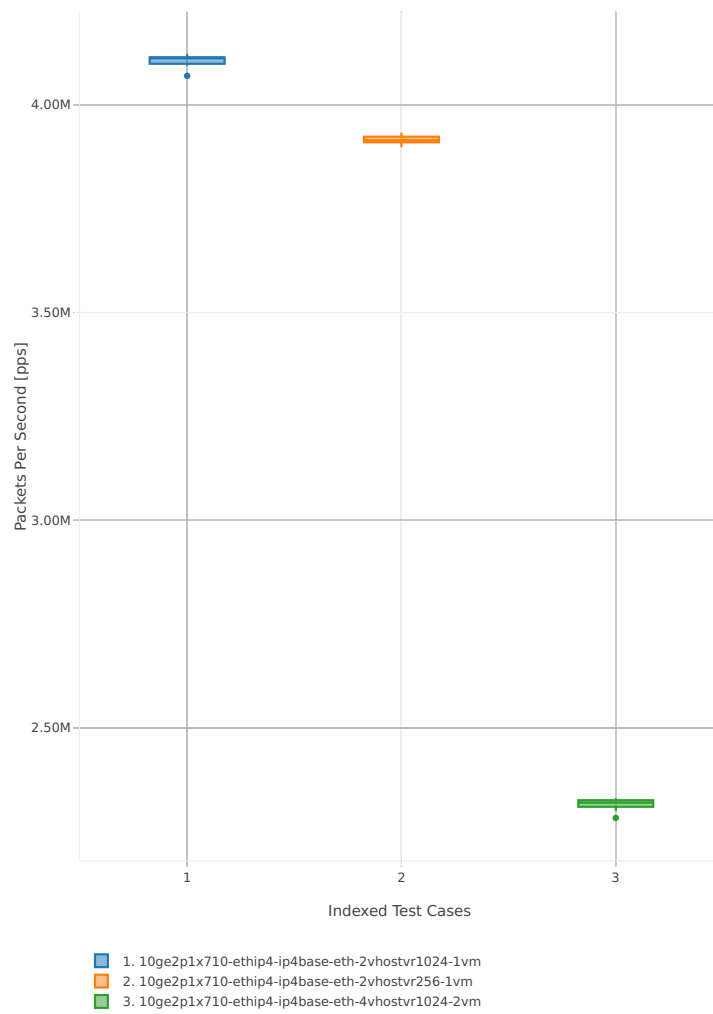


64b-1t1c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

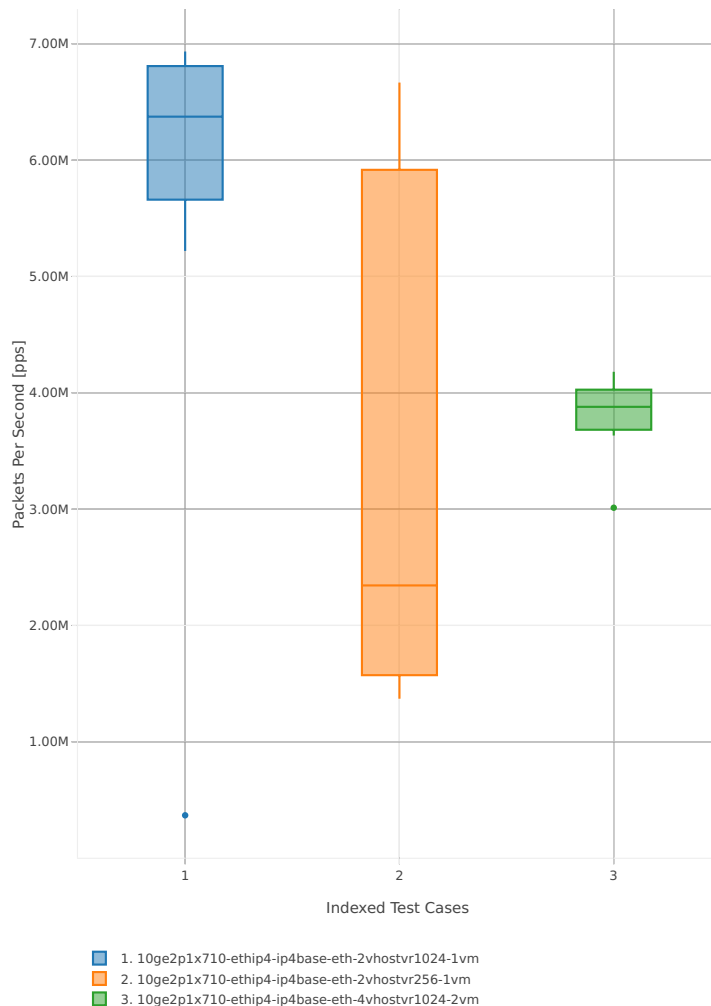


Throughput: vhost-ip4-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

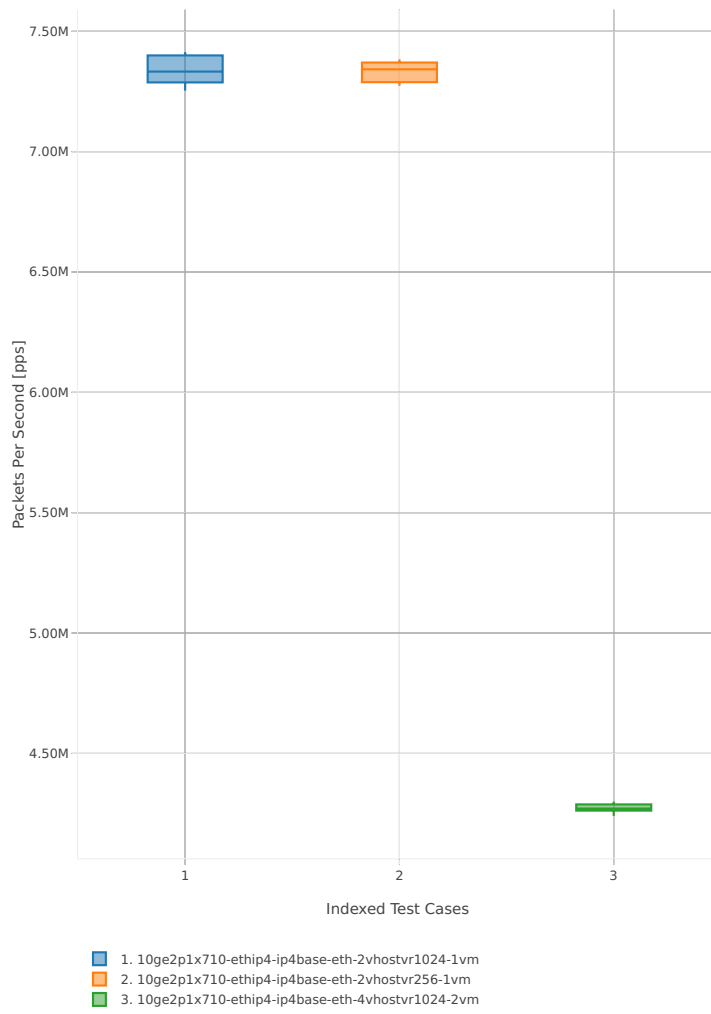


64b-2t2c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



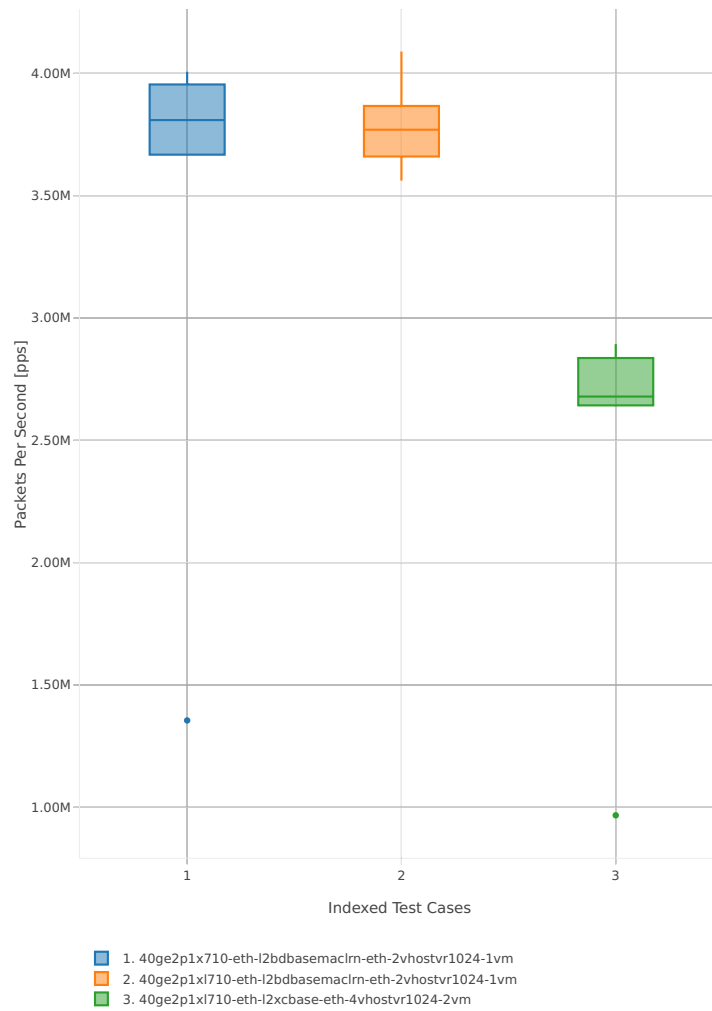
Throughput: vhost-ip4-3n-hsw-x710-64b-2t2c-base_and_scale-pdr



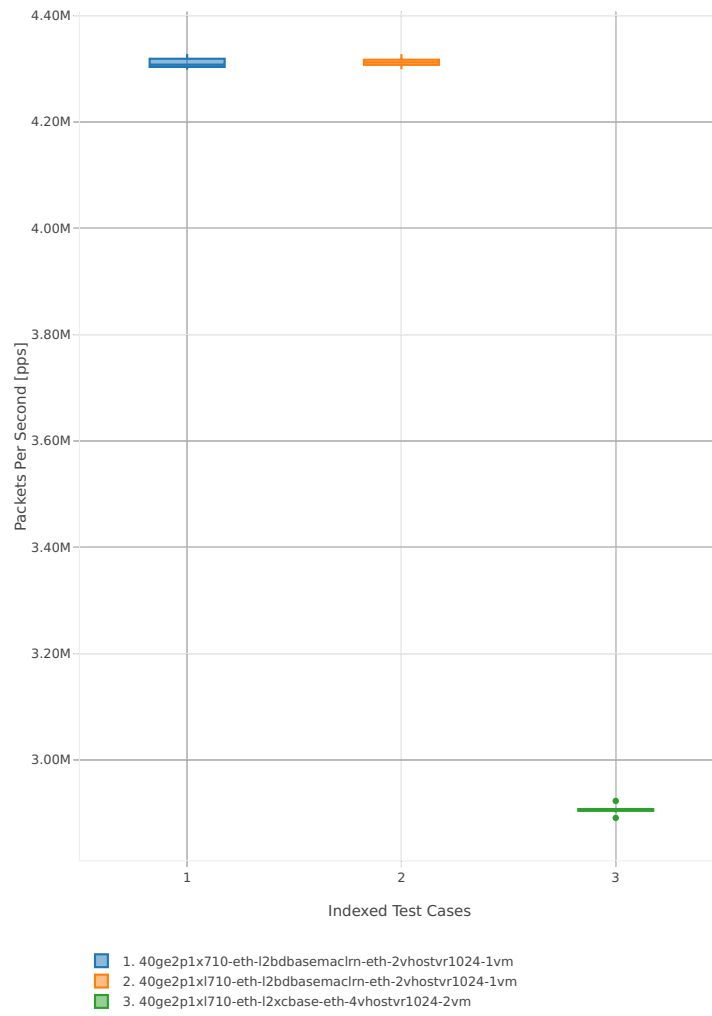
3n-hsw-xl710

64b-1t1c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr

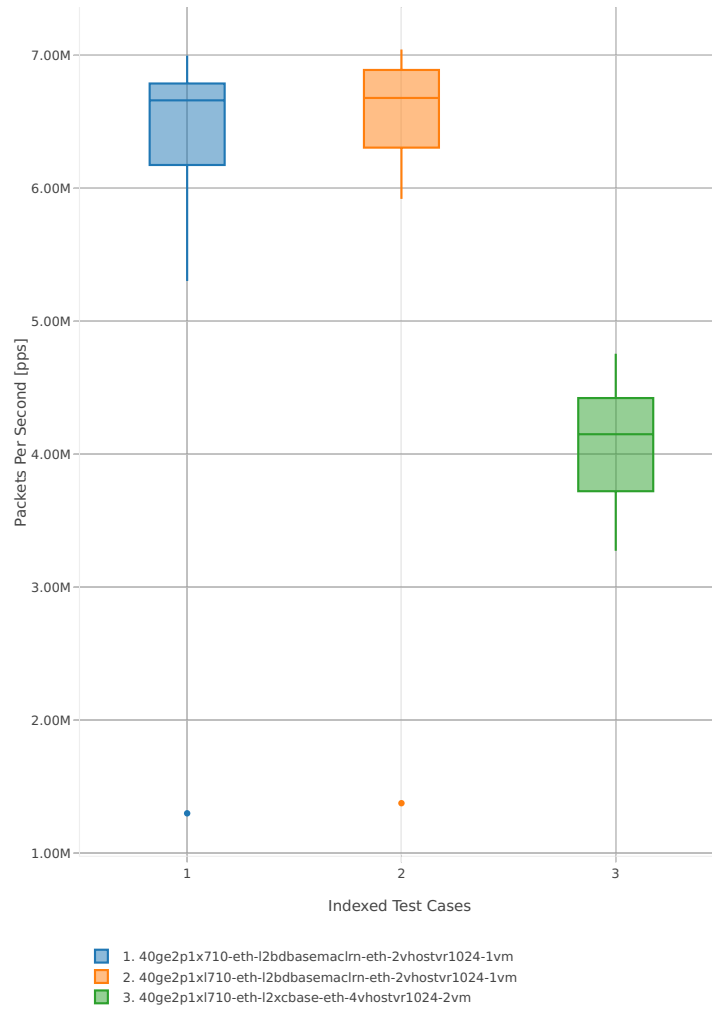


Throughput: vhost-l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

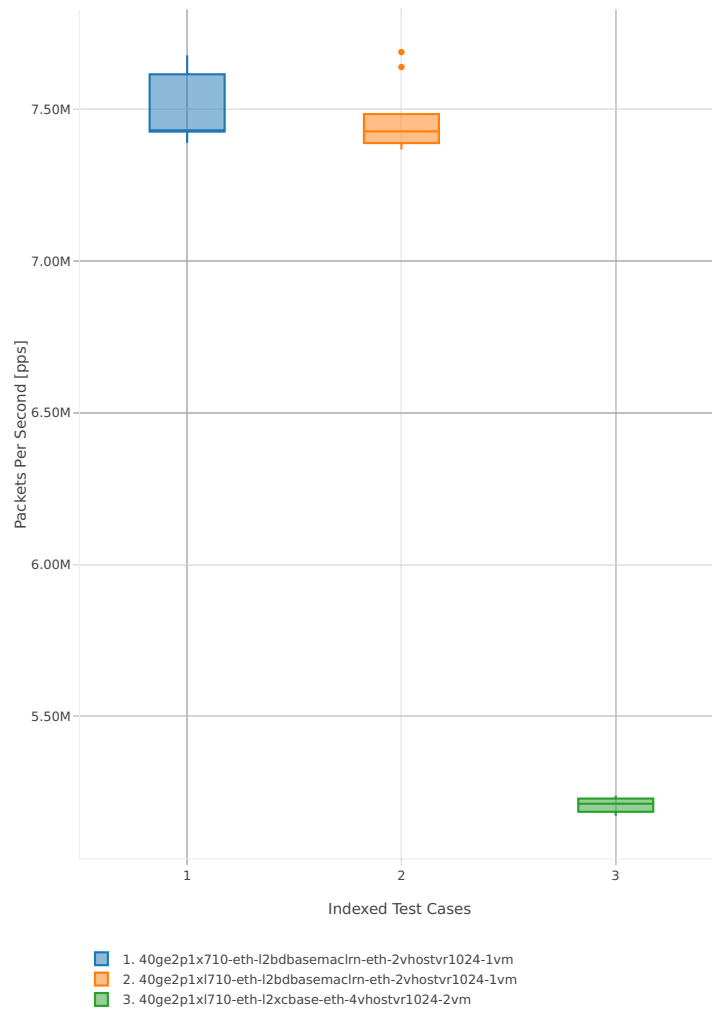


64b-2t2c-base_and_scale-l2sw

Throughput: vhost-l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr

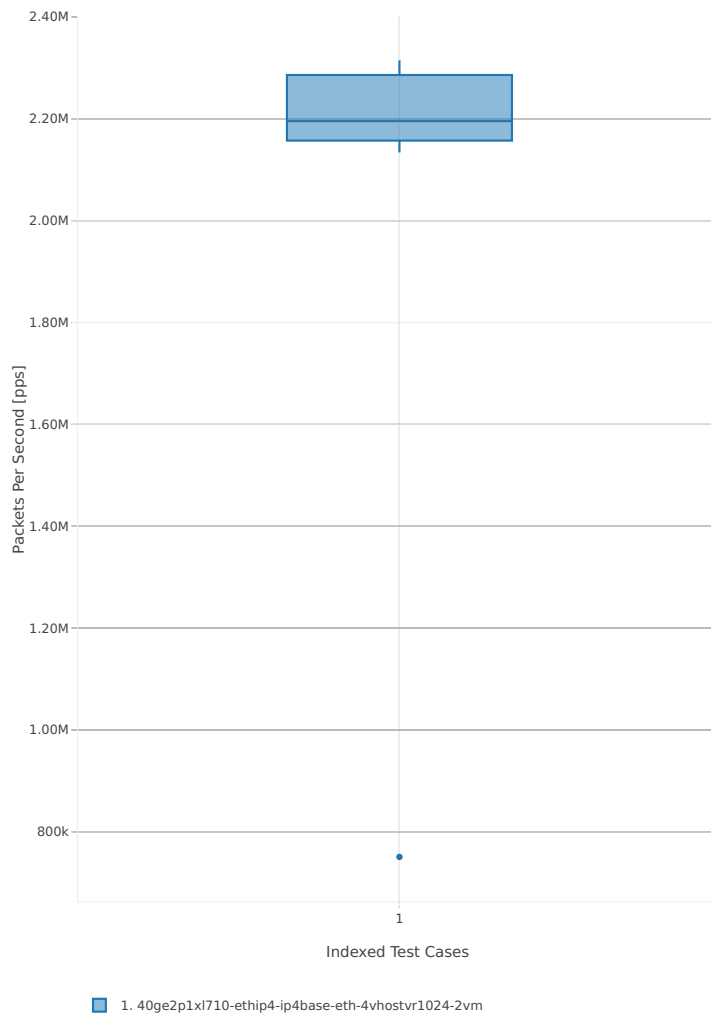


Throughput: vhost-l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr

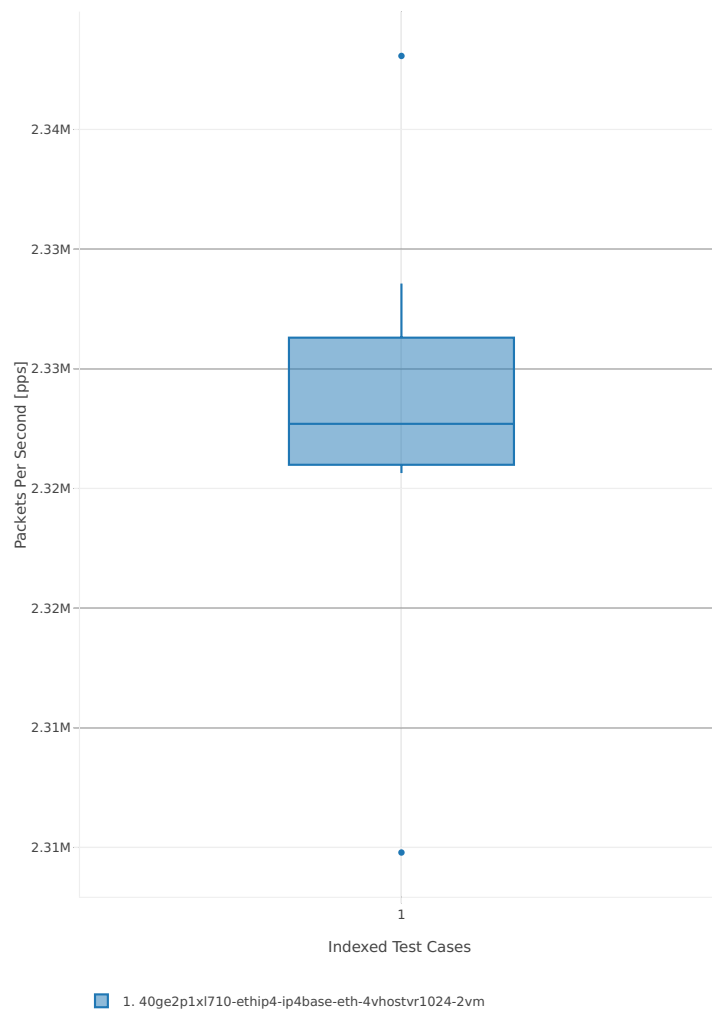


64b-1t1c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr

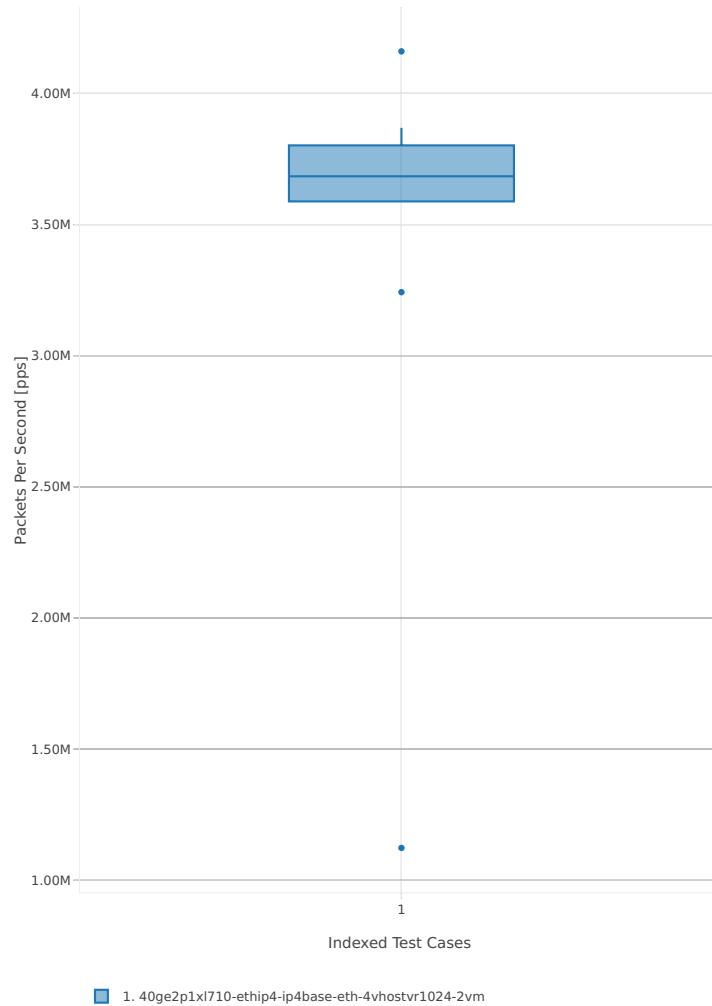


Throughput: vhost-ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

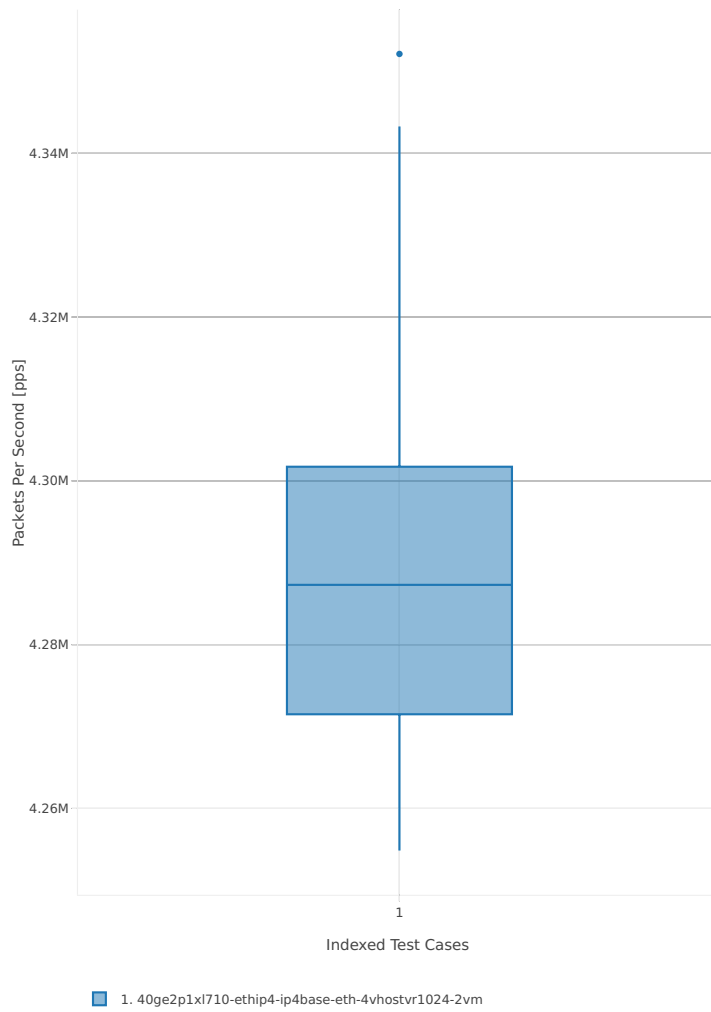


64b-2t2c-base_and_scale-ip4

Throughput: vhost-ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



Throughput: vhost-ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr



2.3.8 LXC/DRC Container Memif

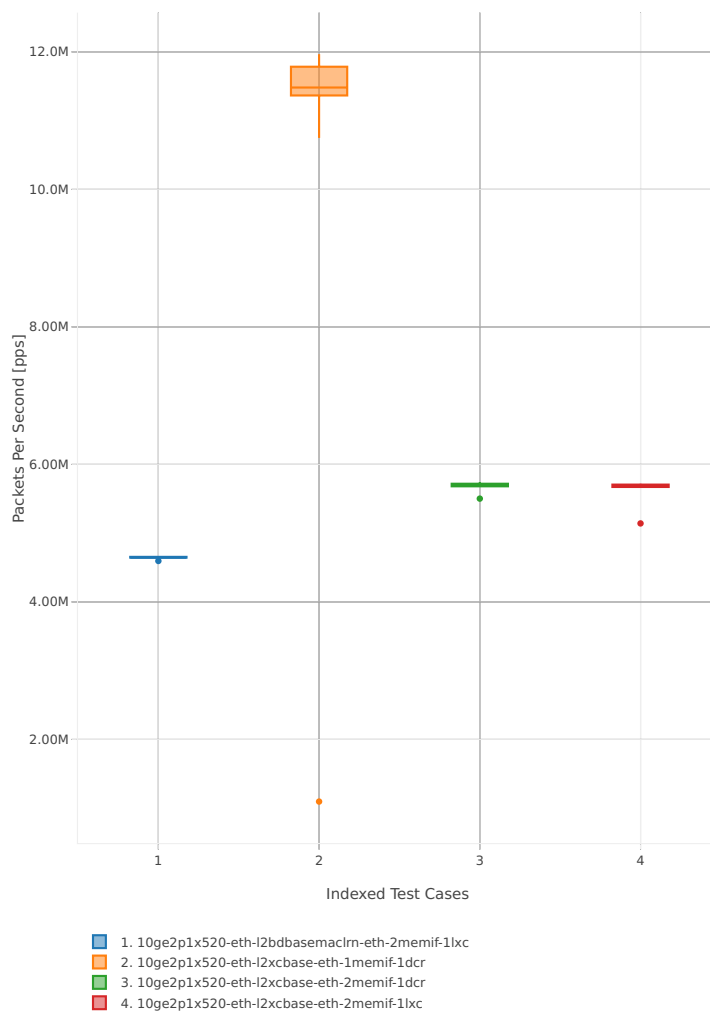
Following sections include summary graphs of VPP Phy-to-Phy performance with Container memif Connections, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane 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](#)³⁶.

3n-hsw-x520

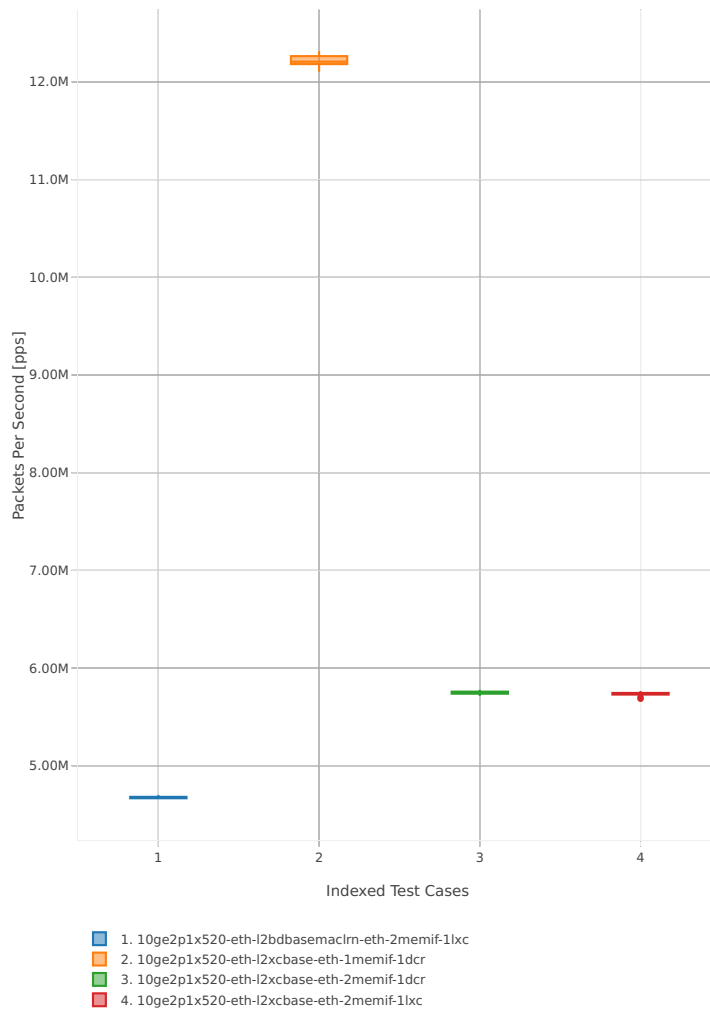
64b-1t1c-base_and_scale

Throughput: memif-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



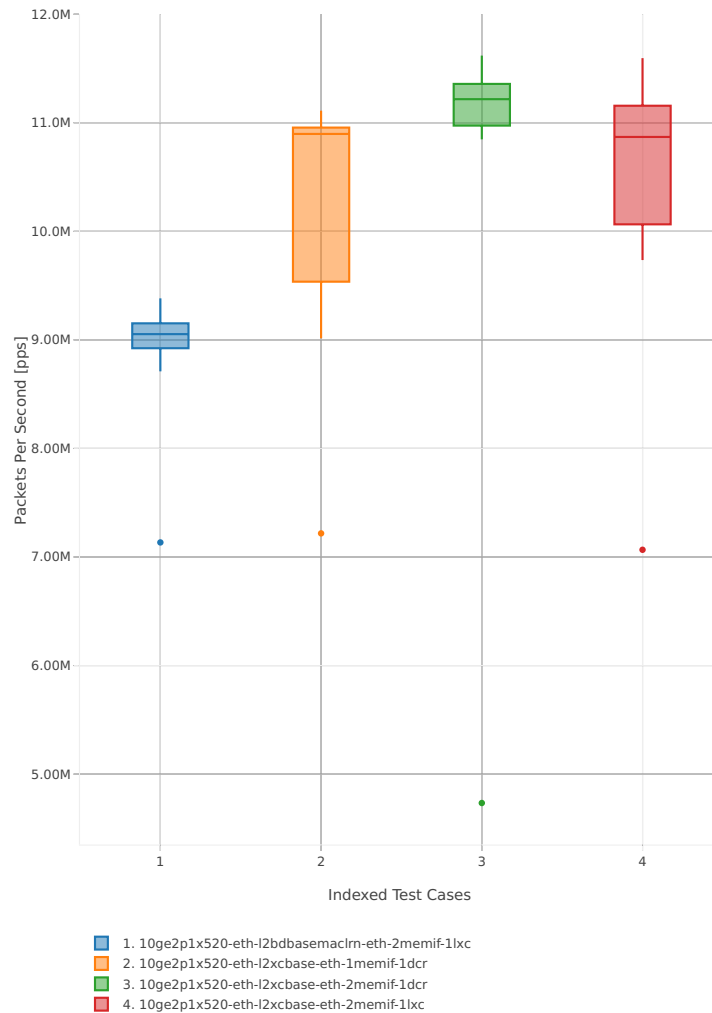
³⁶ https://git.fd.io/csit/tree/tests/vpp/perf/container_memif?h=rls1807

Throughput: memif-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

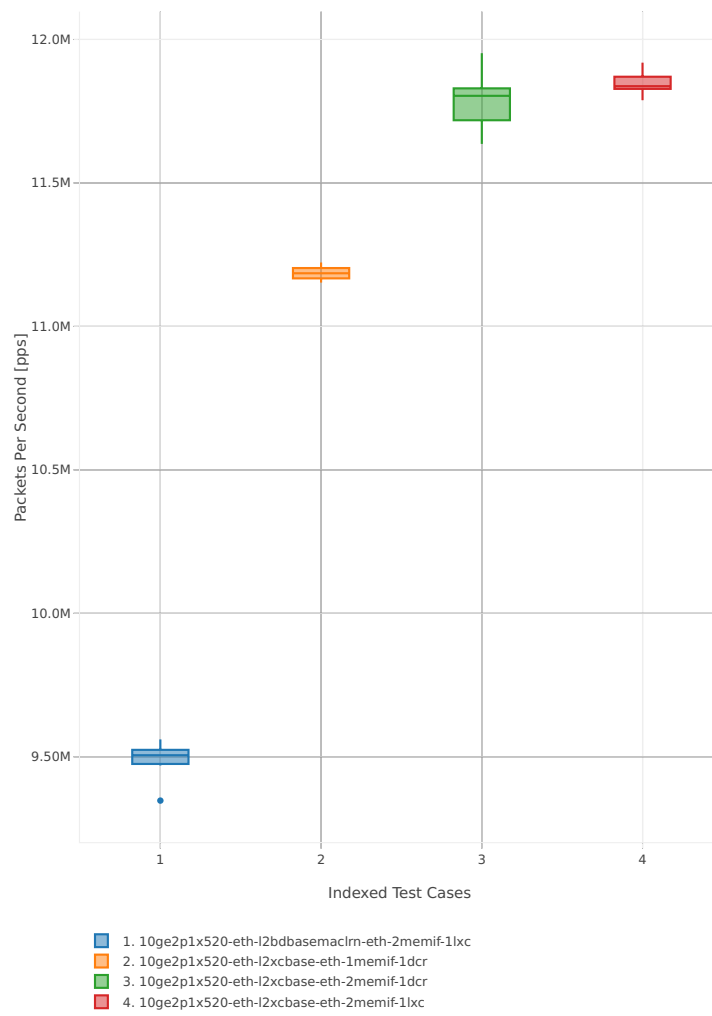


64b-2t2c-base_and_scale

Throughput: memif-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



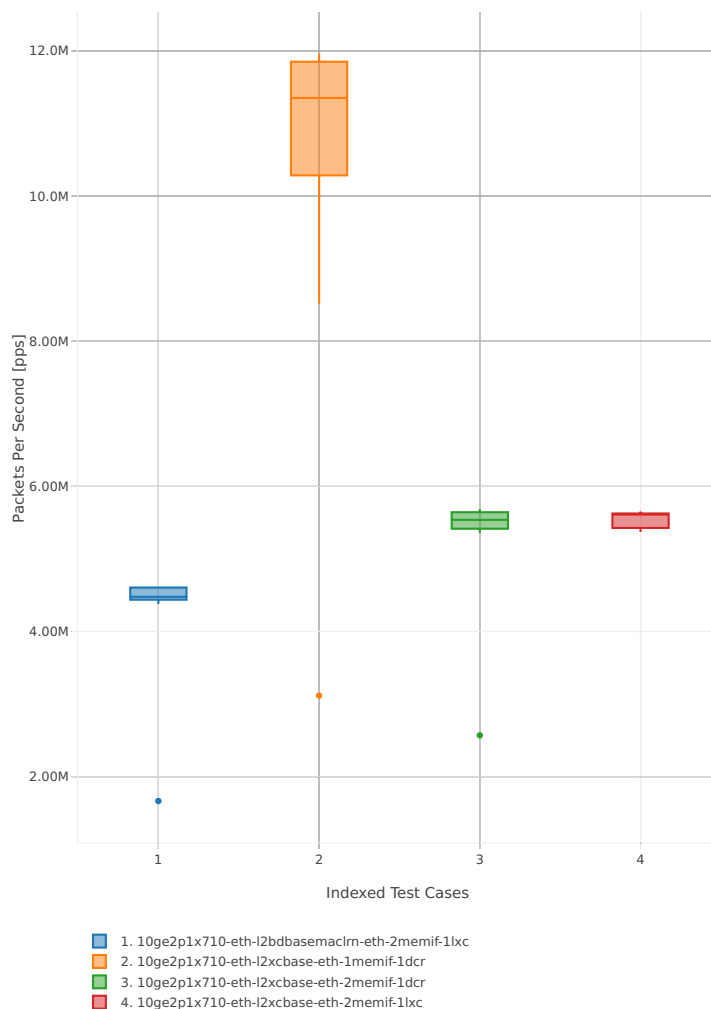
Throughput: memif-3n-hsw-x520-64b-2t2c-base_and_scale-pdr



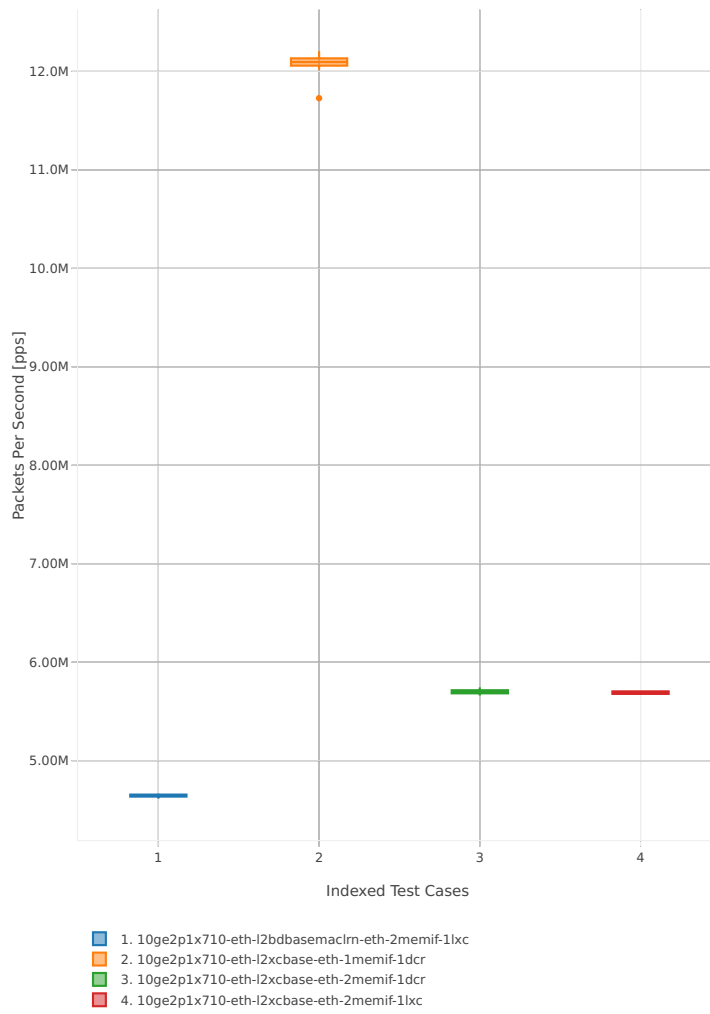
3n-hsw-x710

64b-1t1c-base_and_scale

Throughput: memif-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

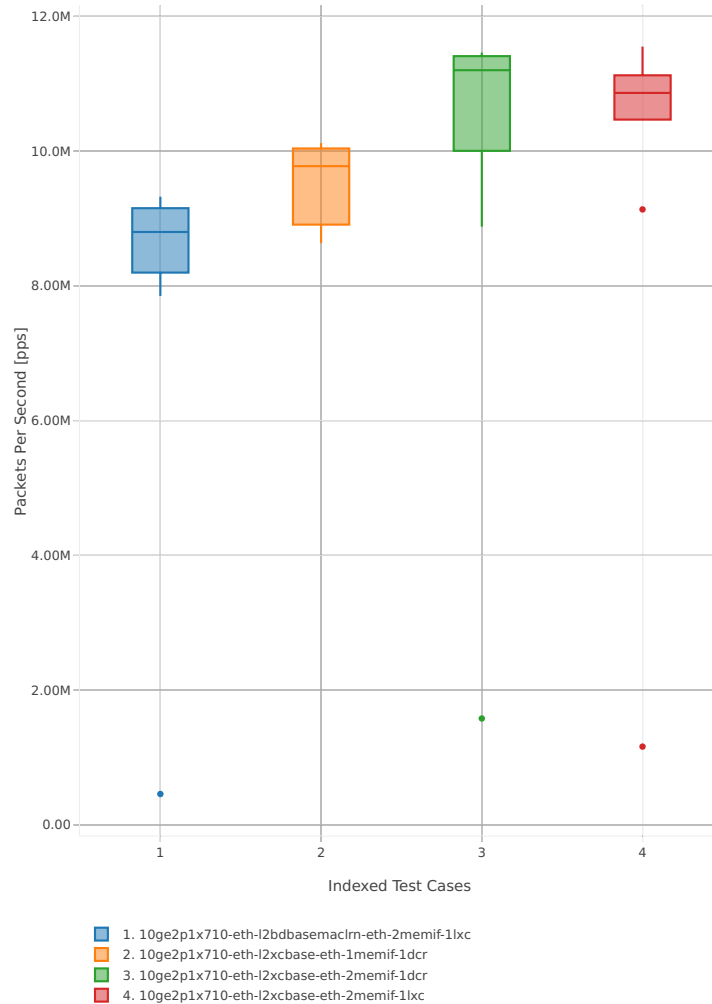


Throughput: memif-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

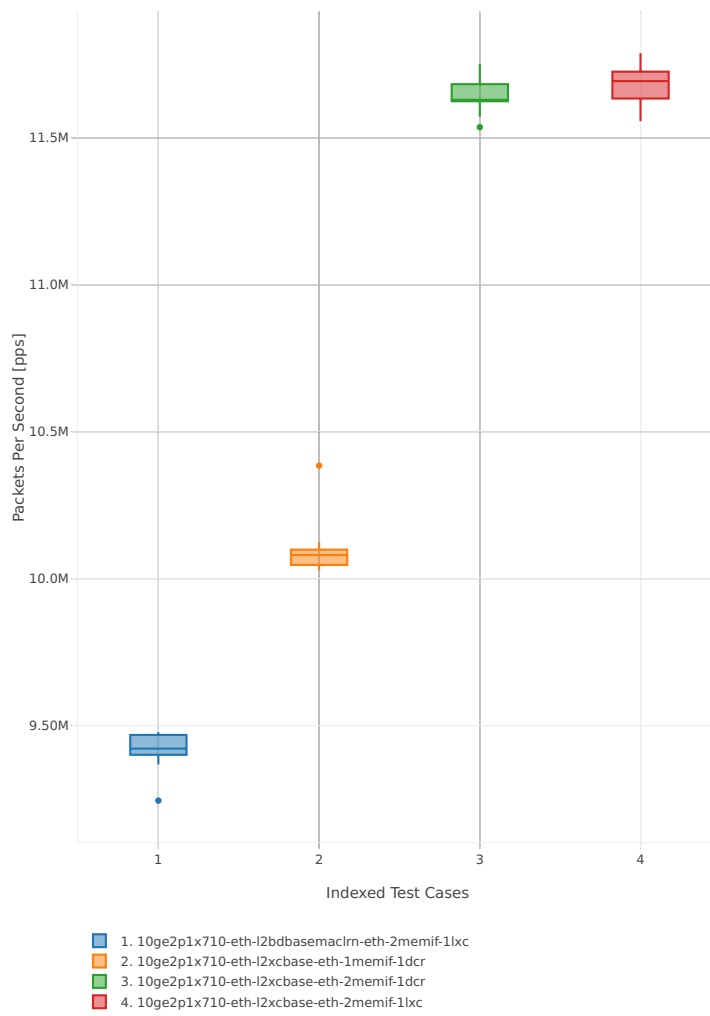


64b-2t2c-base_and_scale

Throughput: memif-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



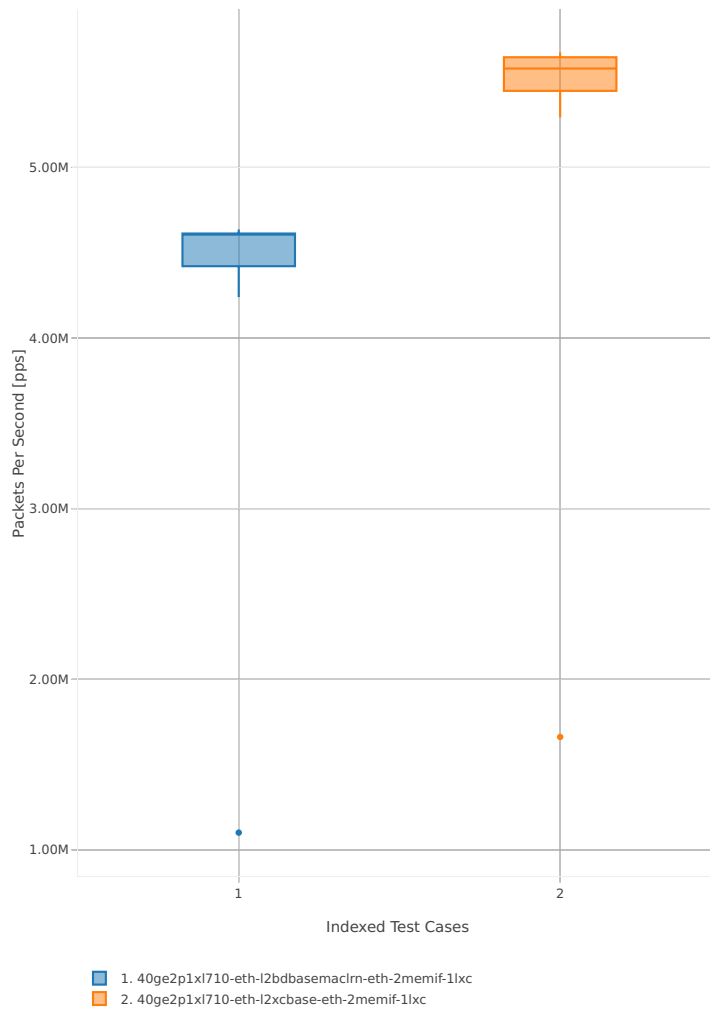
Throughput: memif-3n-hsw-x710-64b-2t2c-base_and_scale-pdr



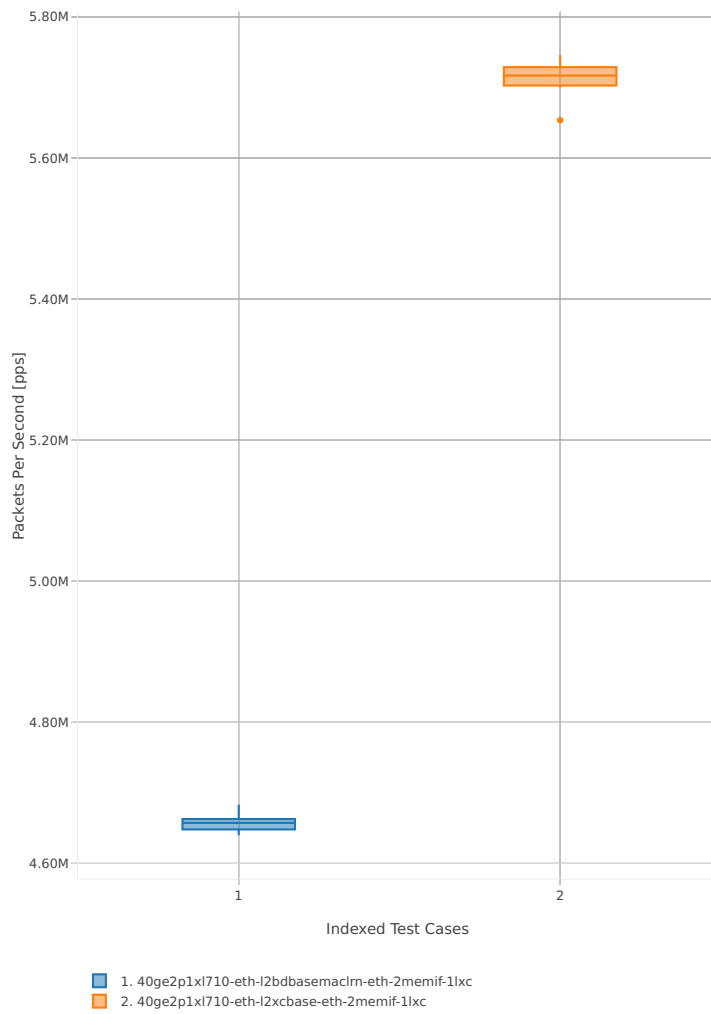
3n-hsw-xl710

64b-1t1c-base_and_scale

Throughput: memif-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr

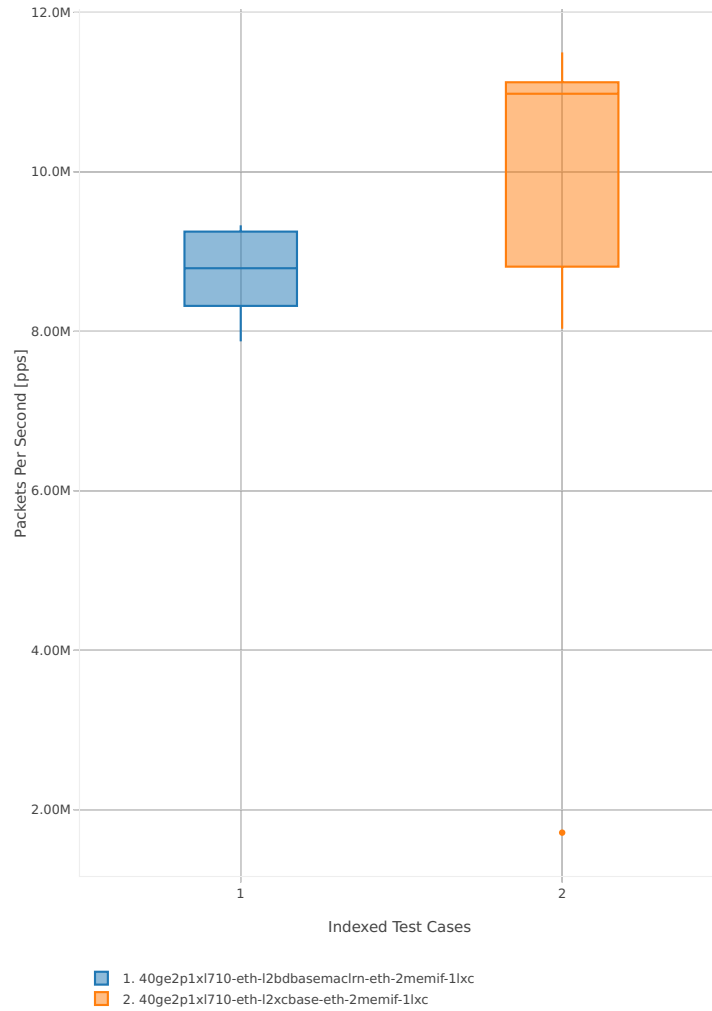


Throughput: memif-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

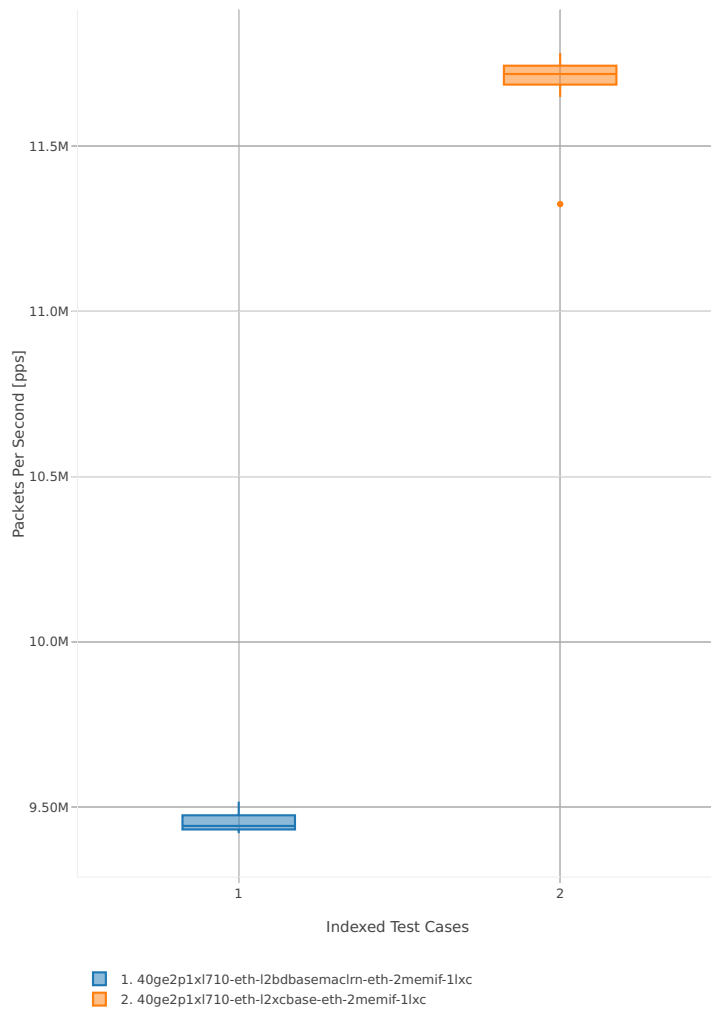


64b-2t2c-base_and_scale

Throughput: memif-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



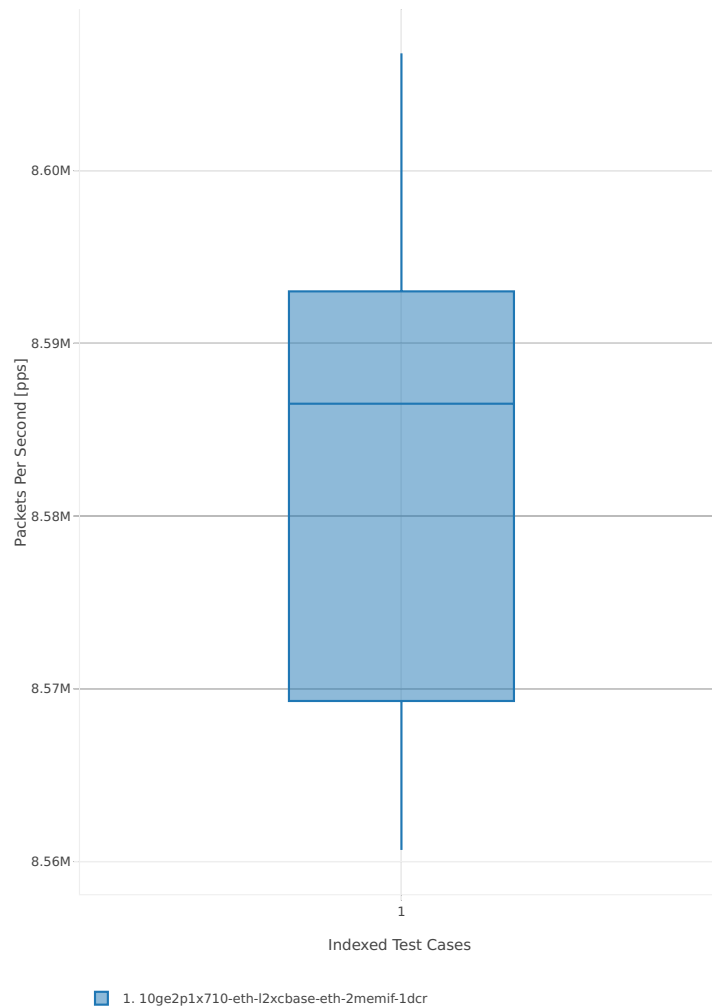
Throughput: memif-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr



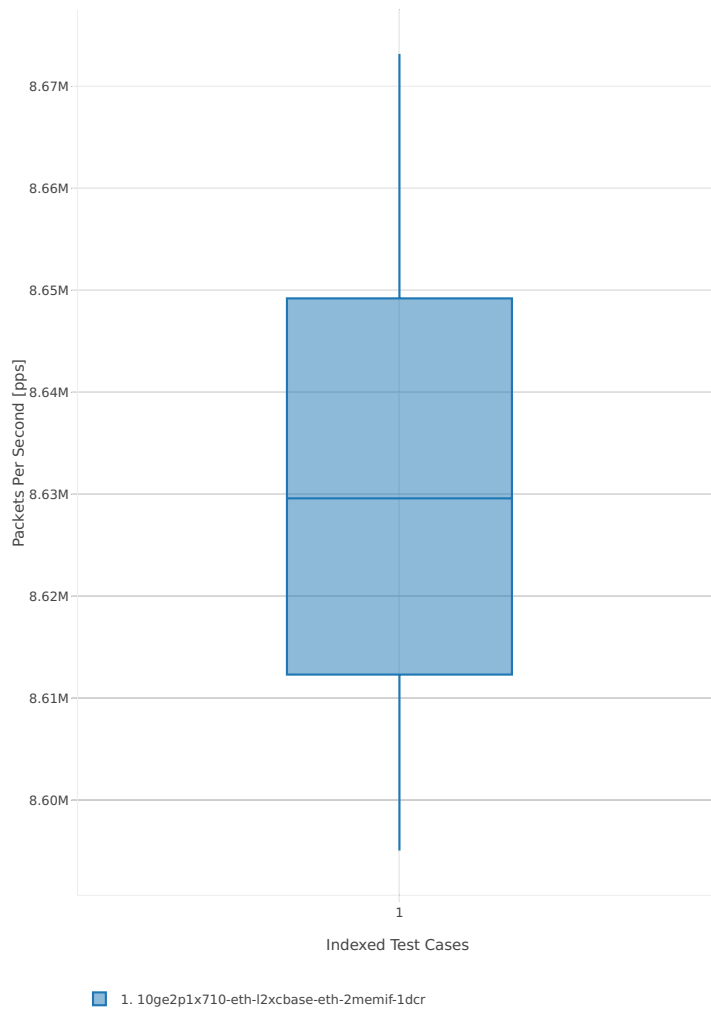
3n-skx-x710

64b-2t1c-base_and_scale

Throughput: memif-3n-skx-x710-64b-2t1c-base_and_scale-ndr

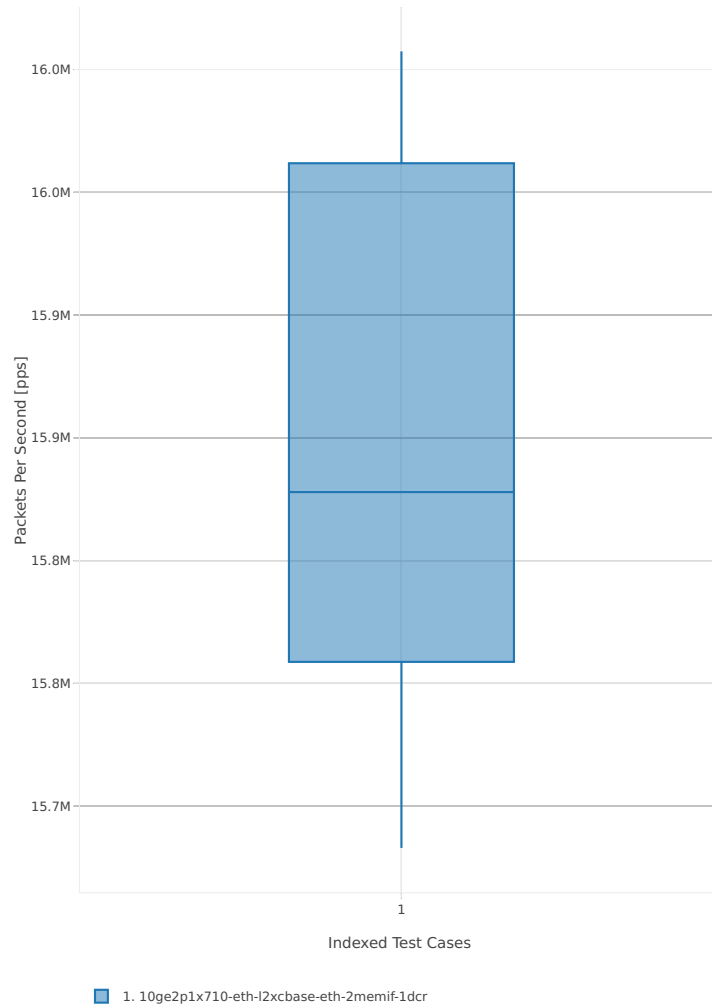


Throughput: memif-3n-skx-x710-64b-2t1c-base_and_scale-pdr

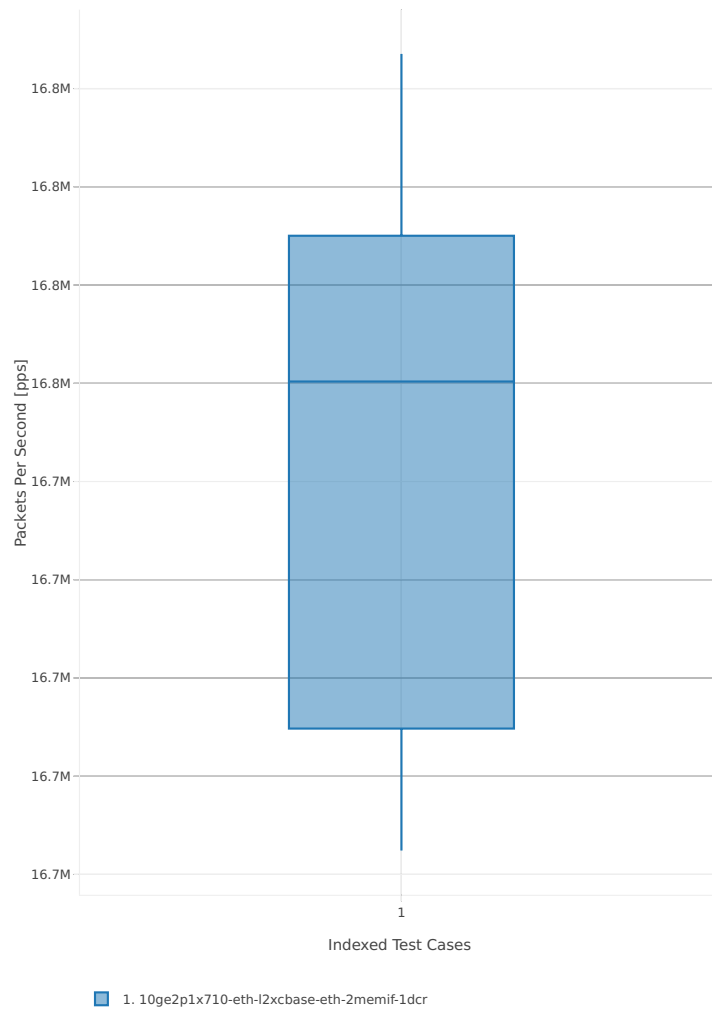


64b-4t2c-base_and_scale

Throughput: memif-3n-skx-x710-64b-4t2c-base_and_scale-ndr



Throughput: memif-3n-skx-x710-64b-4t2c-base_and_scale-pdr



2.3.9 K8s Container Memif

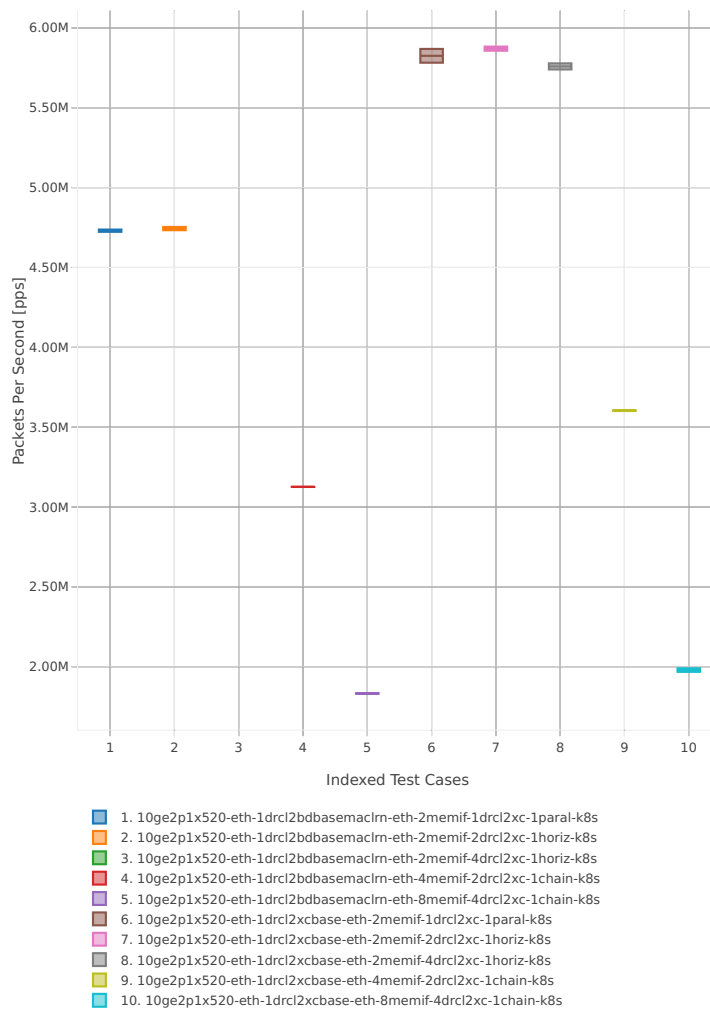
Following sections include summary graphs of VPP Phy-to-Phy performance with Container Orchestrated Topologies, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data 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](#)³⁷.

3n-hsw-x520

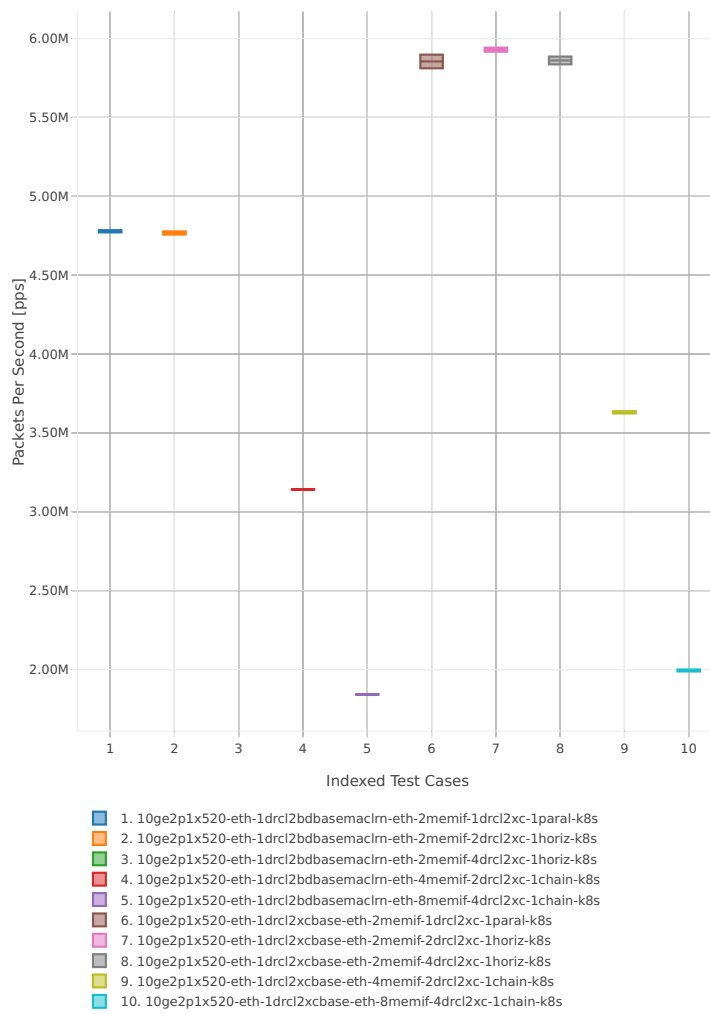
64b-1t1c-base_and_scale

Throughput: k8s-memif-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



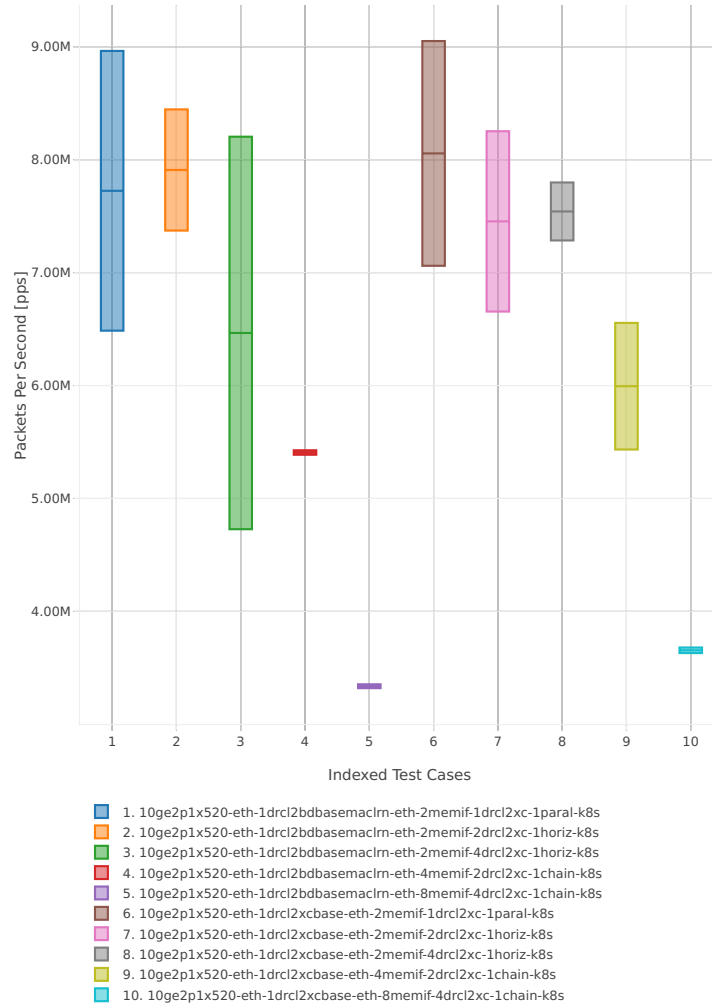
³⁷ https://git.fd.io/csit/tree/tests/kubernetes/perf/container_memif?h=rls1807

Throughput: k8s-memif-3n-hsw-x520-64b-1t1c-base_and_scale-pdr

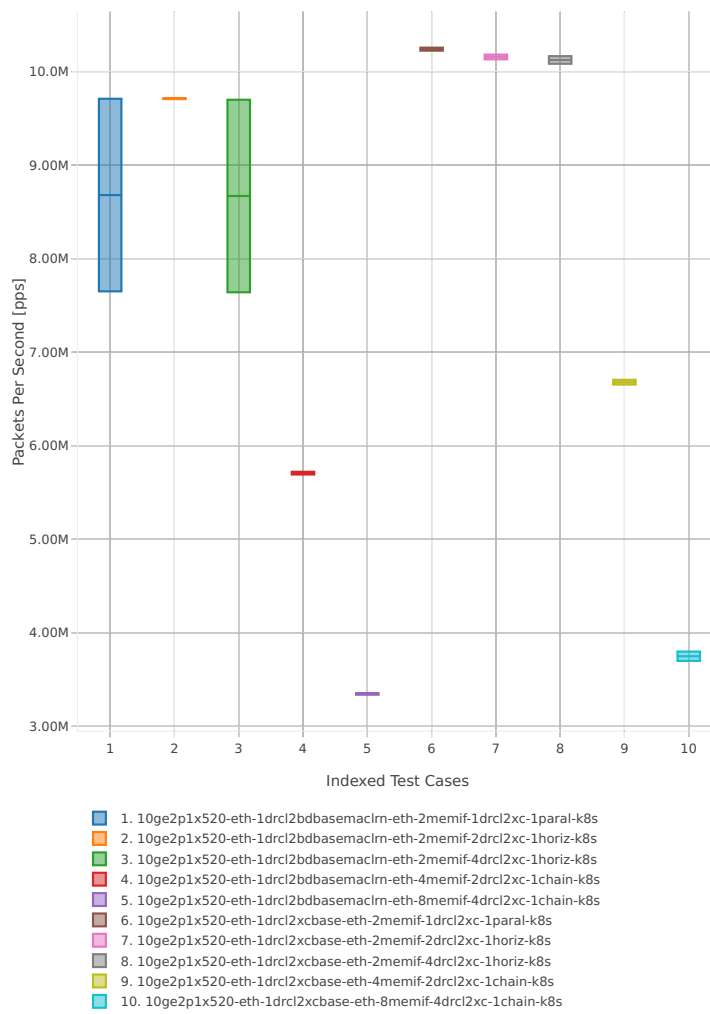


64b-2t2c-base_and_scale

Throughput: k8s-memif-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



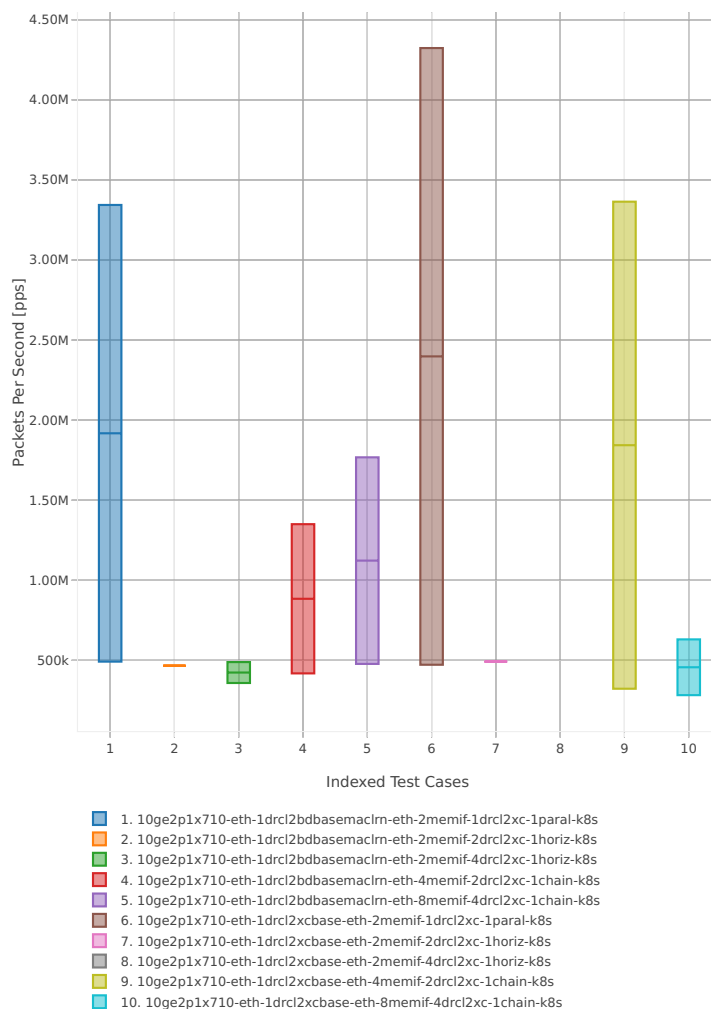
Throughput: k8s-memif-3n-hsw-x520-64b-2t2c-base_and_scale-pdr



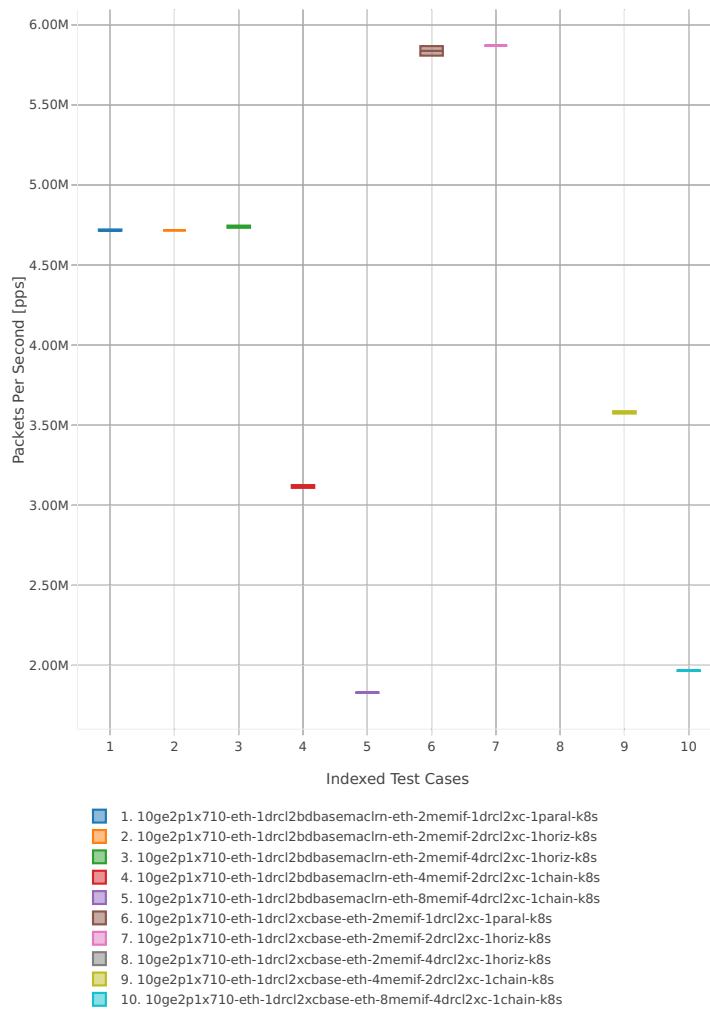
3n-hsw-x710

64b-1t1c-base_and_scale

Throughput: k8s-memif-3n-hsw-x710-64b-1t1c-base_and_scale-ndr

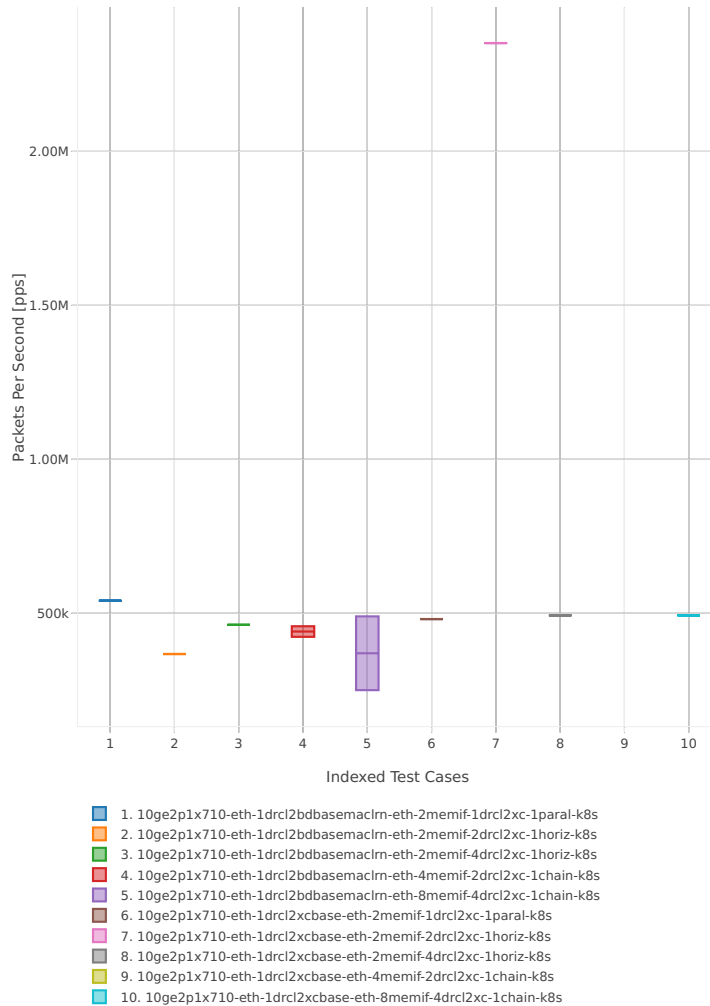


Throughput: k8s-memif-3n-hsw-x710-64b-1t1c-base_and_scale-pdr

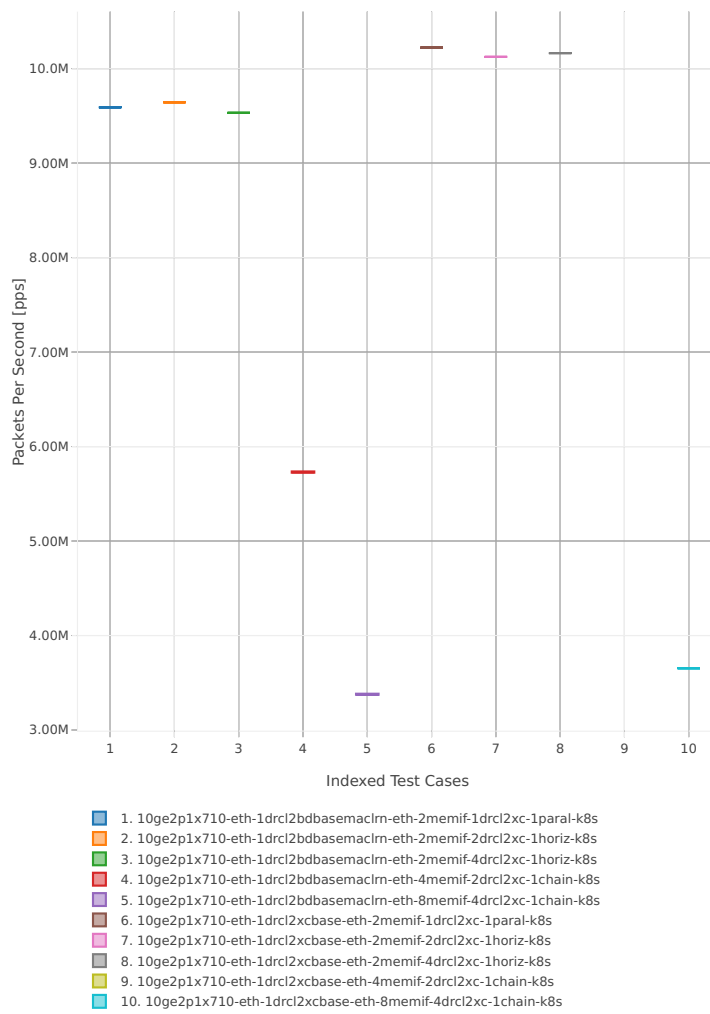


64b-2t2c-base_and_scale

Throughput: k8s-memif-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



Throughput: k8s-memif-3n-hsw-x710-64b-2t2c-base_and_scale-pdr



2.3.10 IPsec IPv4 Routing

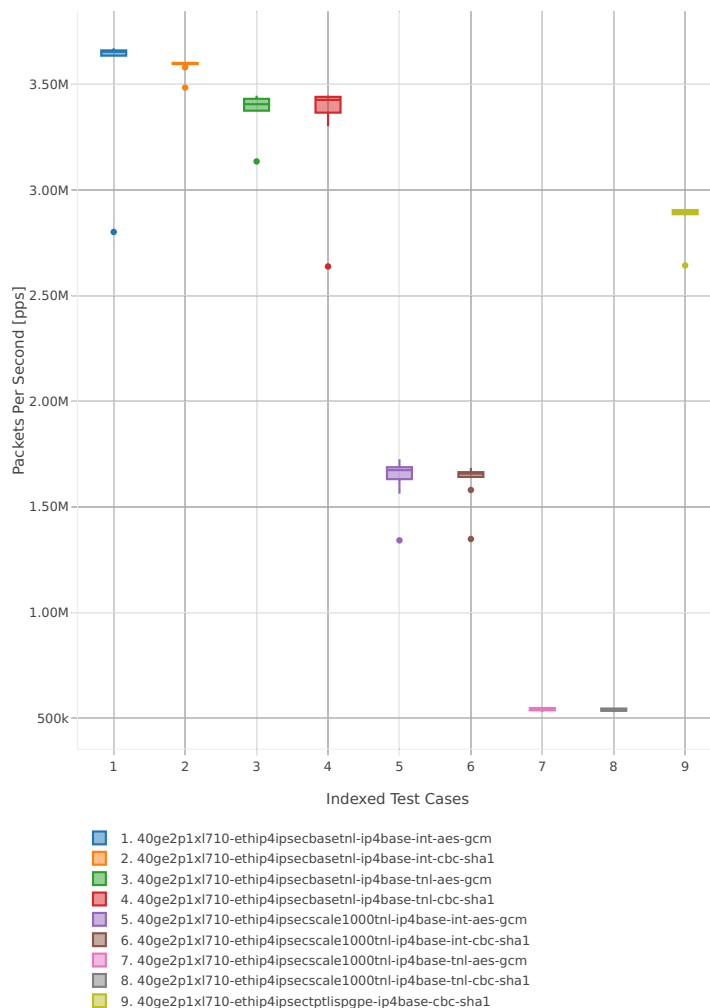
Following sections include summary graphs of VPP Phy-to-Phy performance with IPsec encryption used in combination with IPv4 routed-forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). VPP IPsec encryption is accelerated using DPDK cryptodev library driving Intel 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](#)³⁸.

3n-hsw-xl710

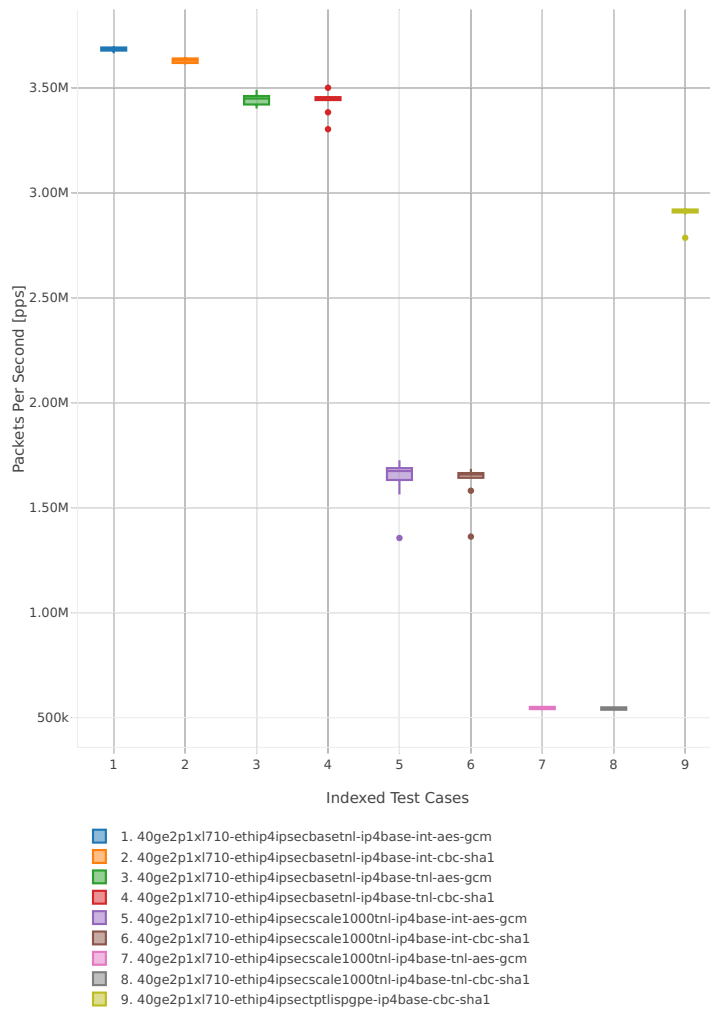
64b-1t1c-base_and_scale

Throughput: ipsec-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



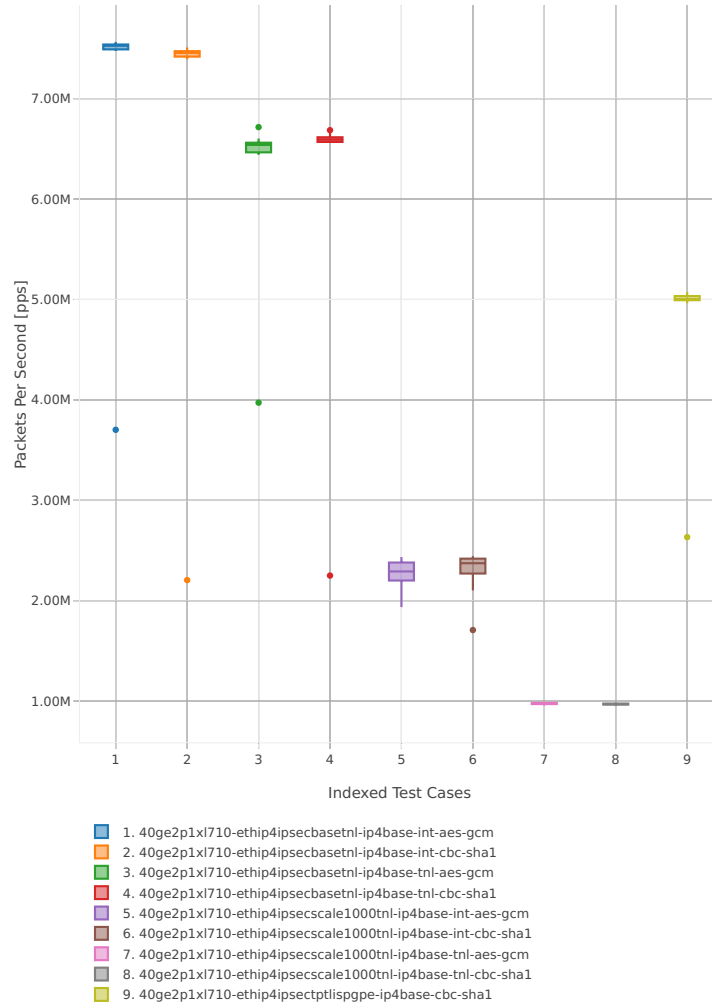
³⁸ <https://git.fd.io/csit/tree/tests/vpp/perf/crypto?h=rls1807>

Throughput: ipsec-3n-hsw-xl710-64b-1t1c-base_and_scale-pdr

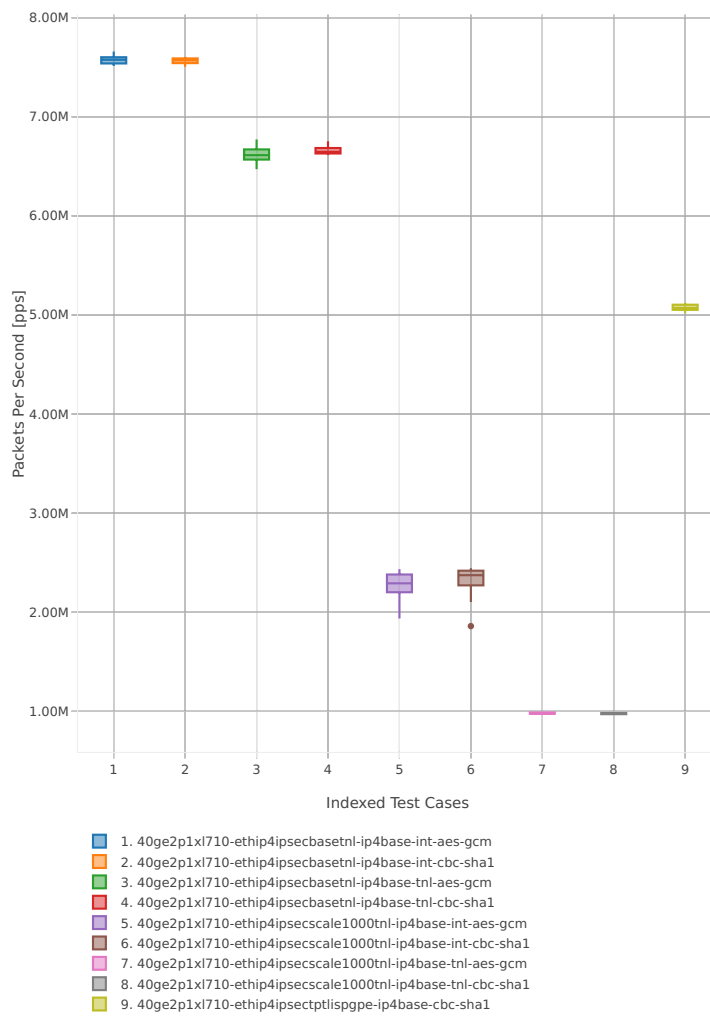


64b-2t2c-base_and_scale

Throughput: ipsec-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



Throughput: ipsec-3n-hsw-xl710-64b-2t2c-base_and_scale-pdr



2.3.11 VTS

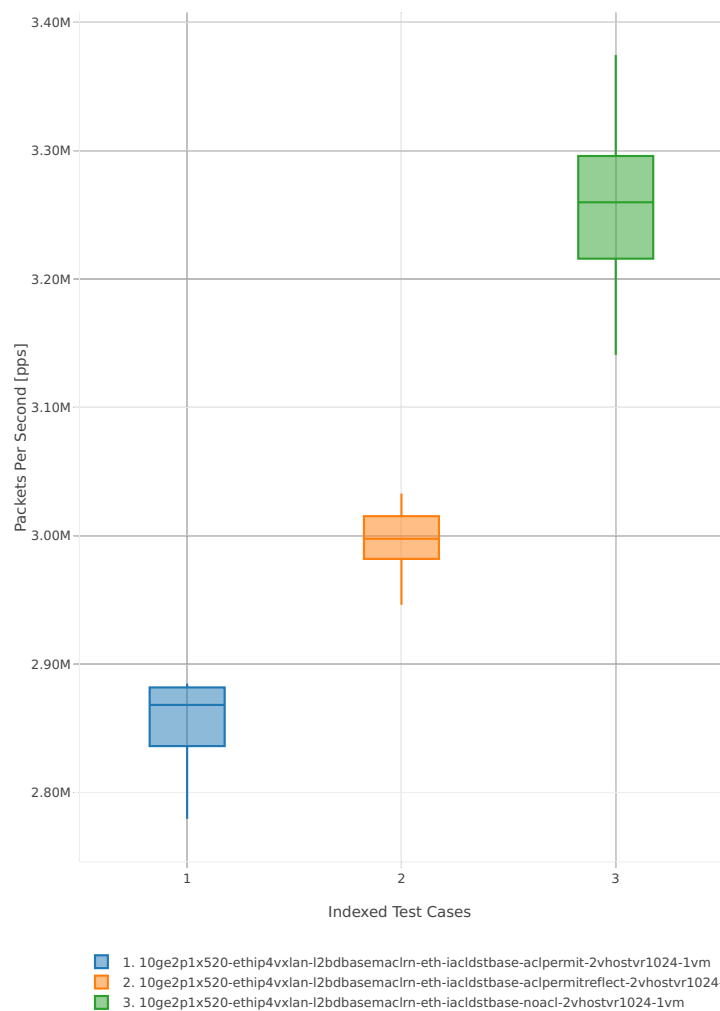
Following sections include summary graphs of VPP Phy-to-VM(s)-to-Phy performance with VM virtio and VPP vhost-user virtual interfaces, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for VPP running in multiple configurations of VPP worker thread(s), 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](#)³⁹.

3n-hsw-x520

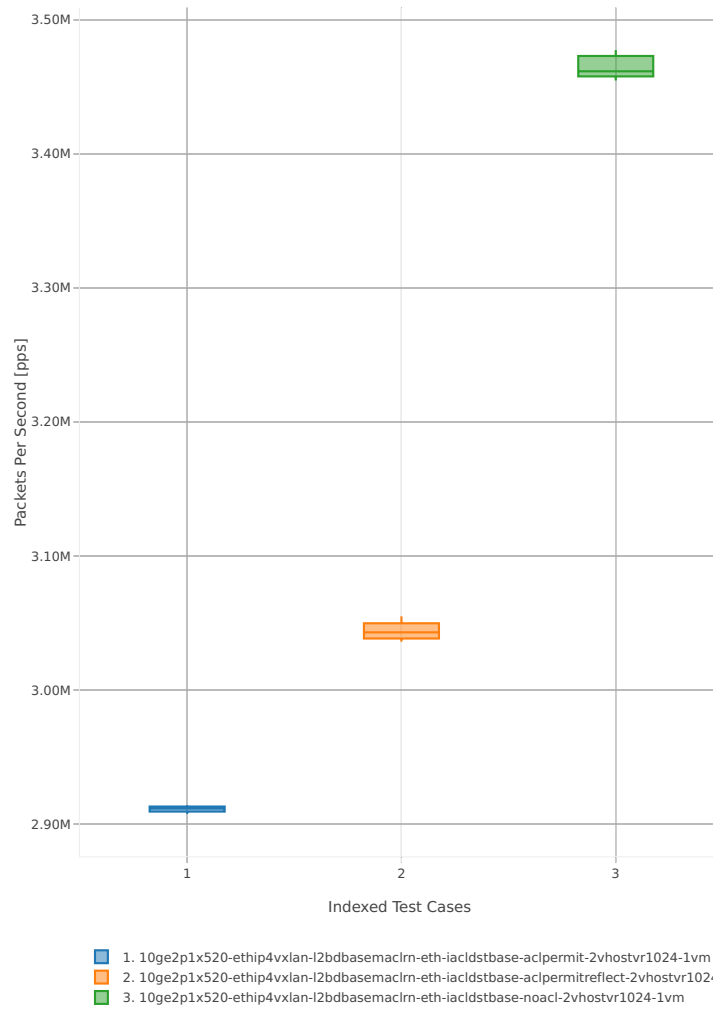
114b-1t1c

Throughput: vts-3n-hsw-x520-114b-1t1c-ndr



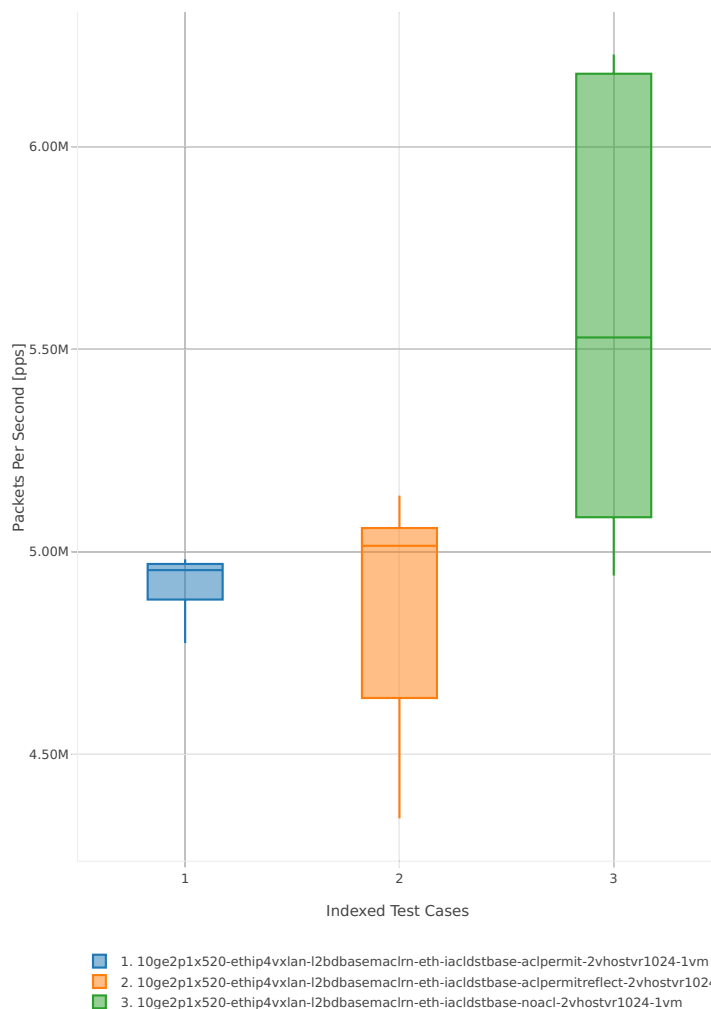
³⁹ <https://git.fd.io/csit/tree/tests/vpp/perf/vts?h=rls1807>

Throughput: vts-3n-hsw-x520-114b-1t1c-pdr

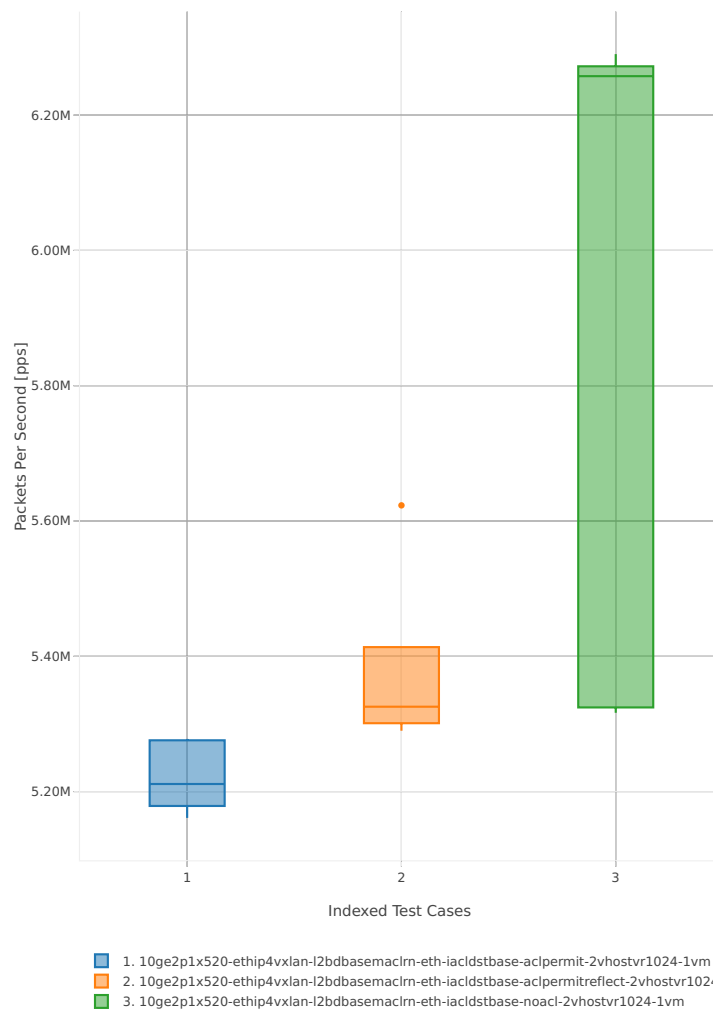


114b-2t2c

Throughput: vts-3n-hsw-x520-114b-2t2c-ndr



Throughput: vts-3n-hsw-x520-114b-2t2c-pdr



2.4 Speedup Multi-Core

Speedup Multi-Core throughput graphs are generated by multiple executions of the same performance tests across physical testbeds hosted LF FD.io labs: 3n-hsw, 2n-skx, 2n-skx. Grouped bars illustrate the 64B/78B packet throughput speedup ratio for 2- and 4-core multi- threaded VPP configurations relative to 1-core configurations.

Note: Test results have been generated by [FD.io test executor vpp performance job 3n-hsw⁴⁰](#), [FD.io test executor vpp performance job 3n-skx⁴¹](#) and [FD.io test executor vpp performance job 2n-skx⁴²](#) with RF result files csit-vpp-perf-1807-*.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 . This is due to not all test executions completing successfully due to a known [VPP-1361⁴³](#) issue sporadically impacting physical interface up/down state.

⁴⁰ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

⁴¹ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-skx>

⁴² <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-2n-skx>

⁴³ <https://jira.fd.io/browse/VPP-1361>

2.4.1 L2 Ethernet Switching

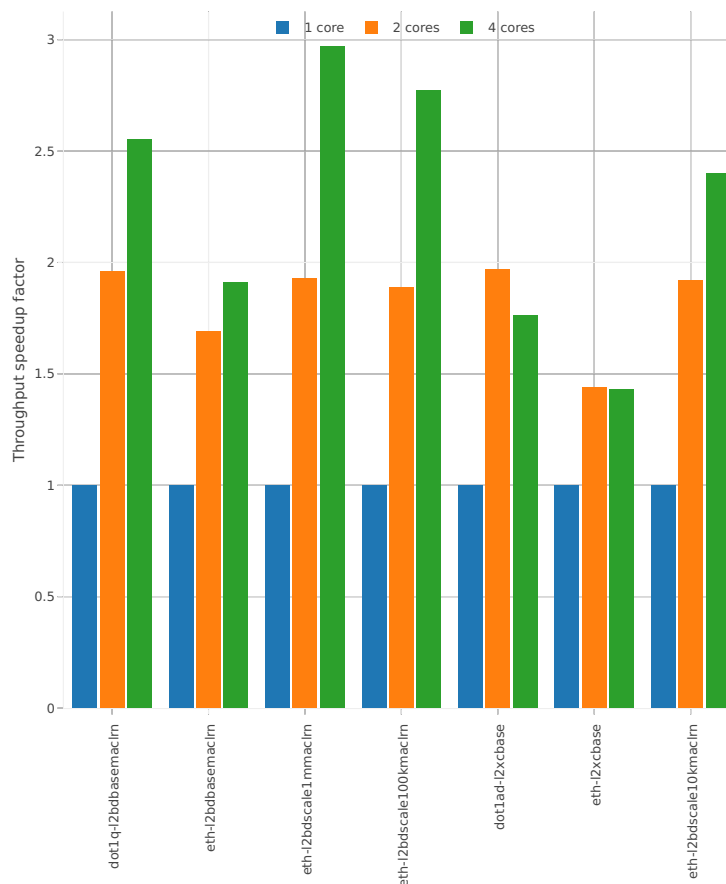
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 64B performance tests with VPP L2 Ethernet switching, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

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

3n-hsw-x520

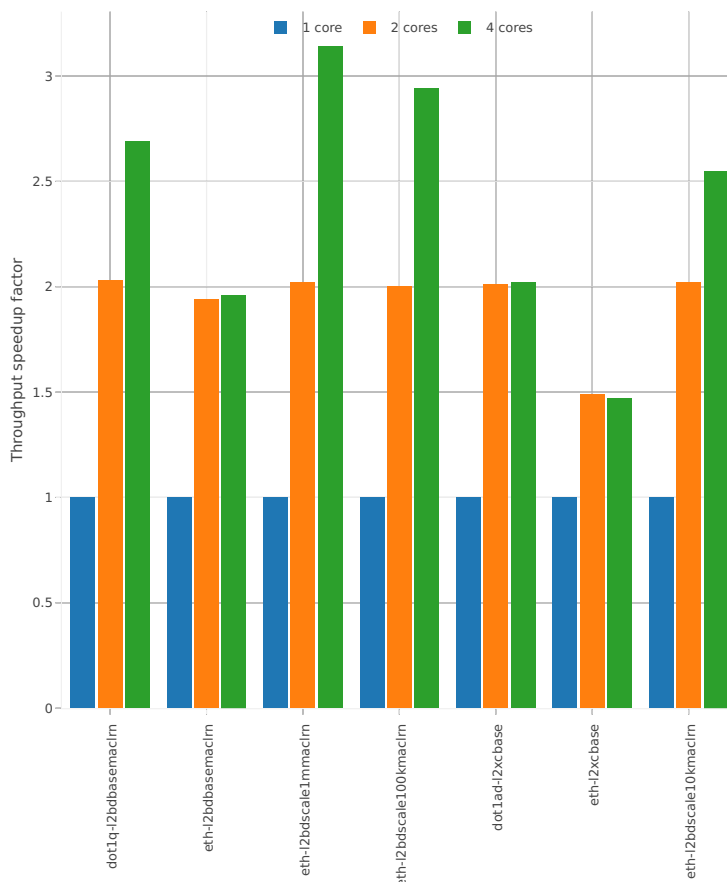
64b-base_and_scale

Speedup: l2sw-3n-hsw-x520-64b-base_and_scale-ndr



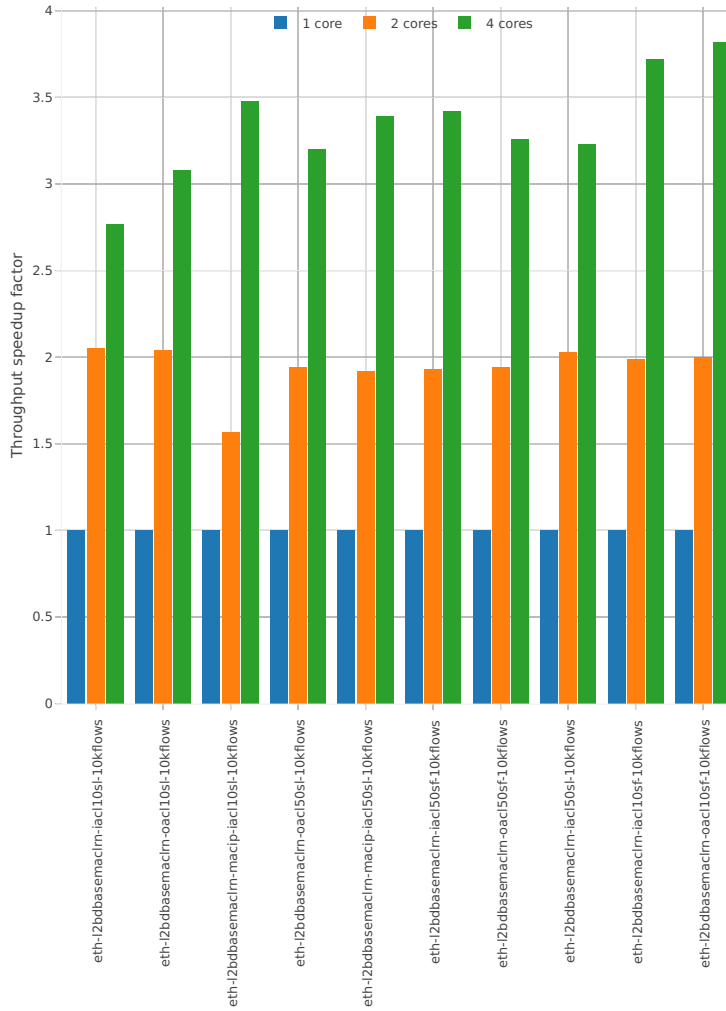
⁴⁴ <https://git.fd.io/csit/tree/tests/vpp/perf/l2?h=rls1807>

Speedup: l2sw-3n-hsw-x520-64b-base_and_scale-pdr

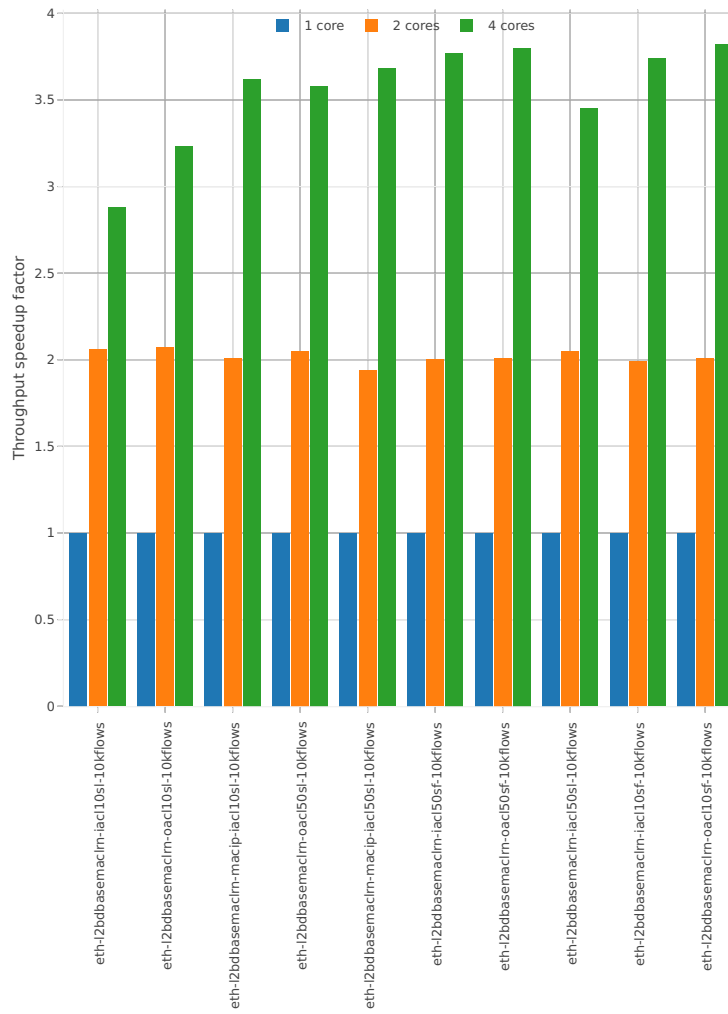


64b-features

Speedup: l2sw-3n-hsw-x520-64b-features-ndr



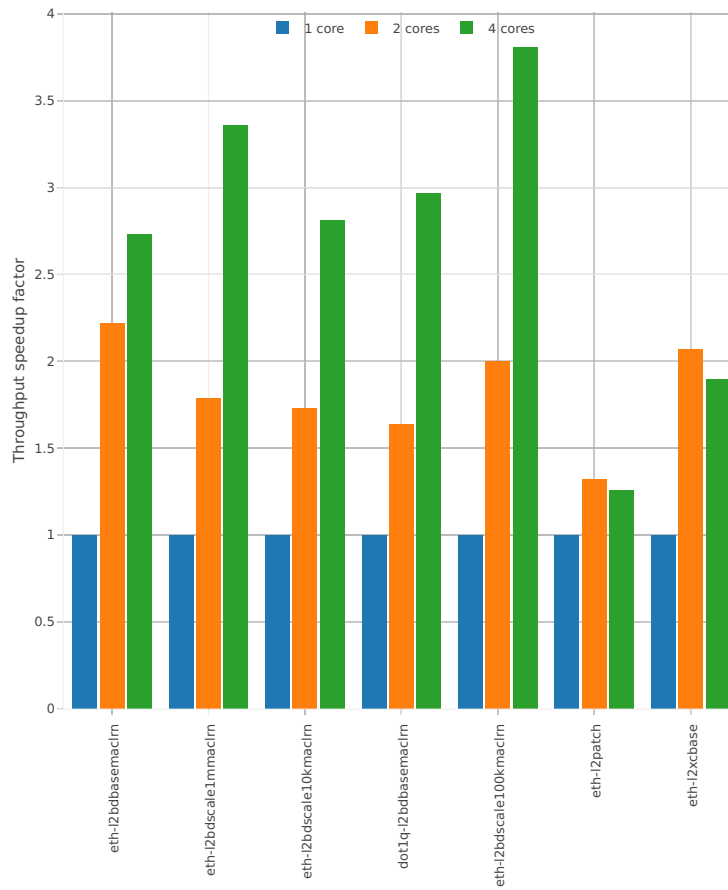
Speedup: l2sw-3n-hsw-x520-64b-features-pdr



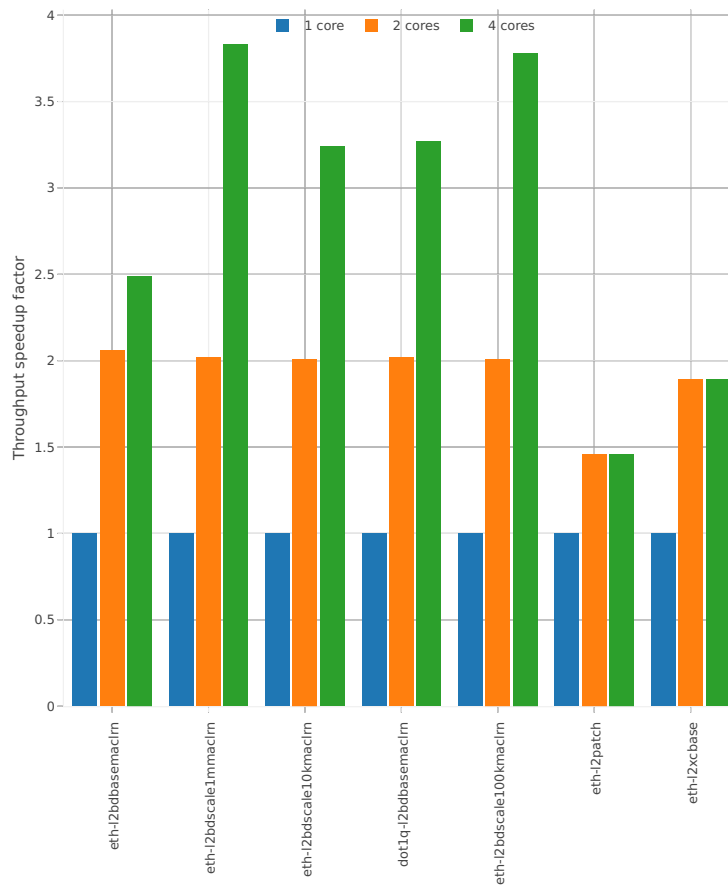
3n-hsw-x710

64b-base_and_scale

Speedup: l2sw-3n-hsw-x710-64b-base_and_scale-ndr

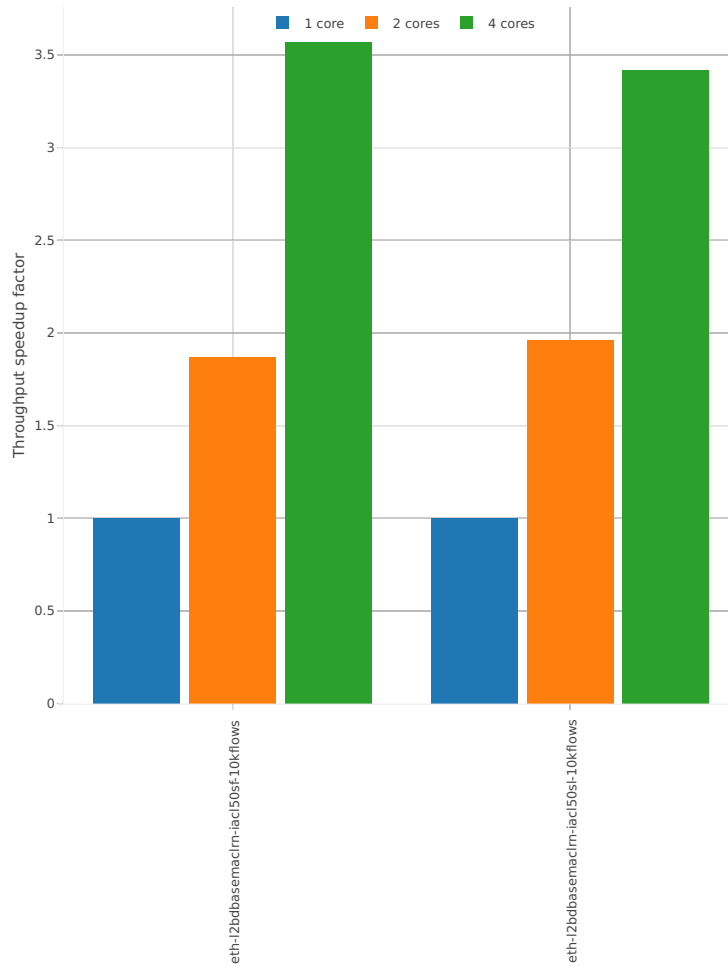


Speedup: l2sw-3n-hsw-x710-64b-base_and_scale-pdr

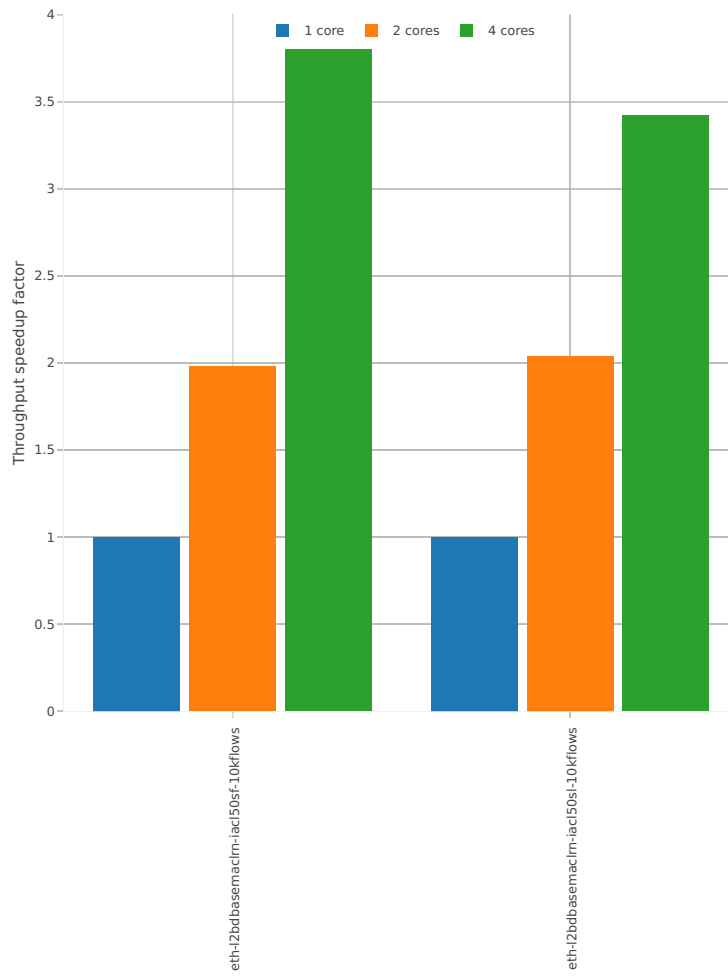


64b-features

Speedup: l2sw-3n-hsw-x710-64b-features-ndr



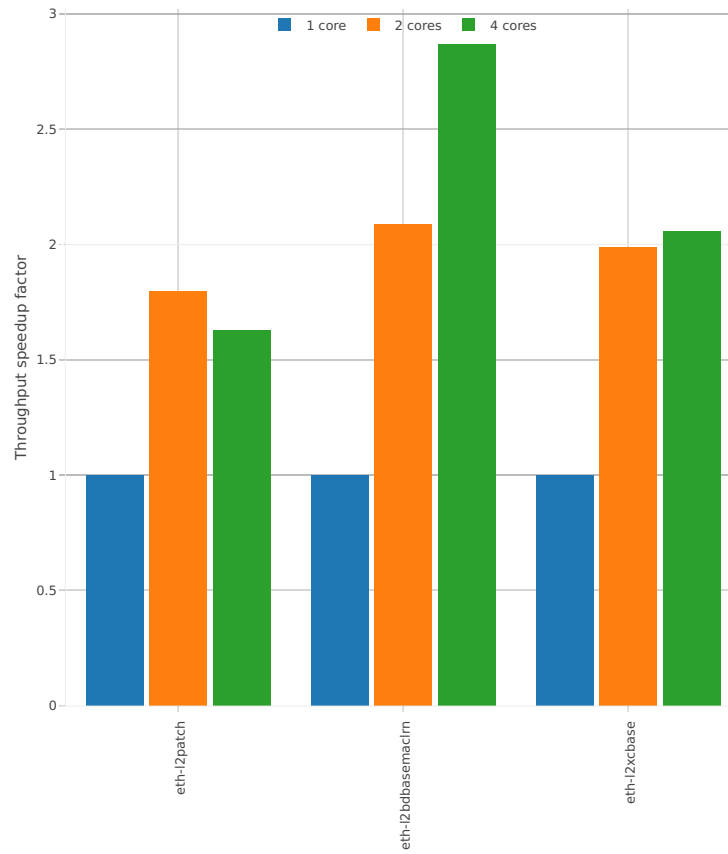
Speedup: l2sw-3n-hsw-x710-64b-features-pdr



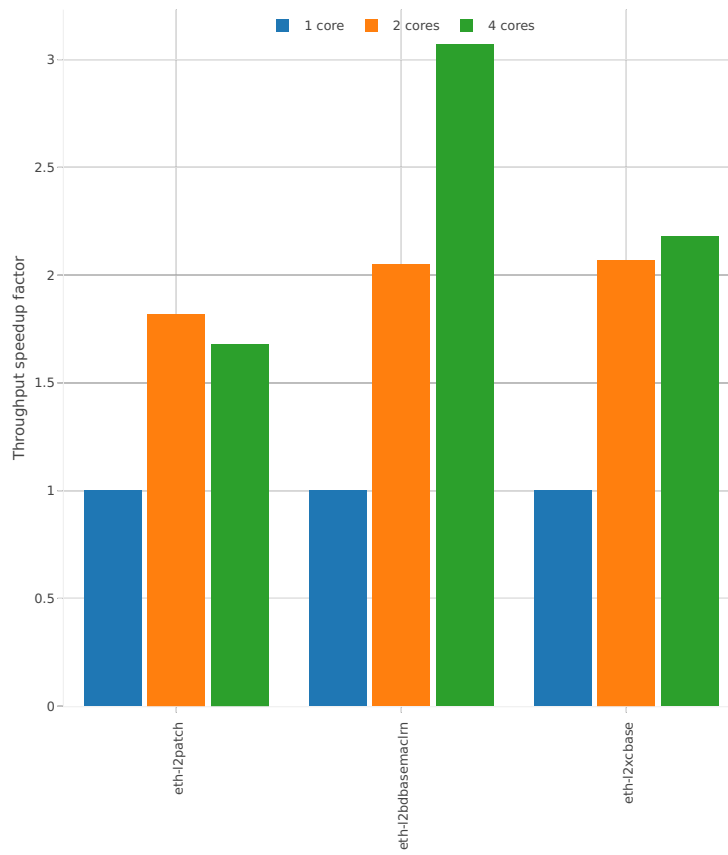
3n-hsw-xl710

64b-base_and_scale

Speedup: l2sw-3n-hsw-xl710-64b-base_and_scale-ndr



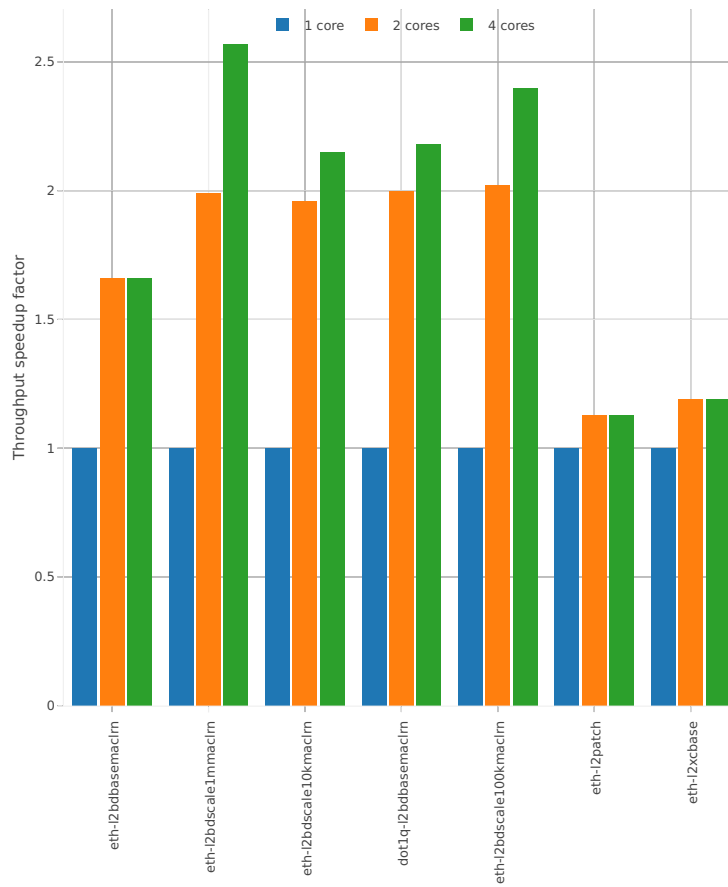
Speedup: l2sw-3n-hsw-xl710-64b-base_and_scale-pdr



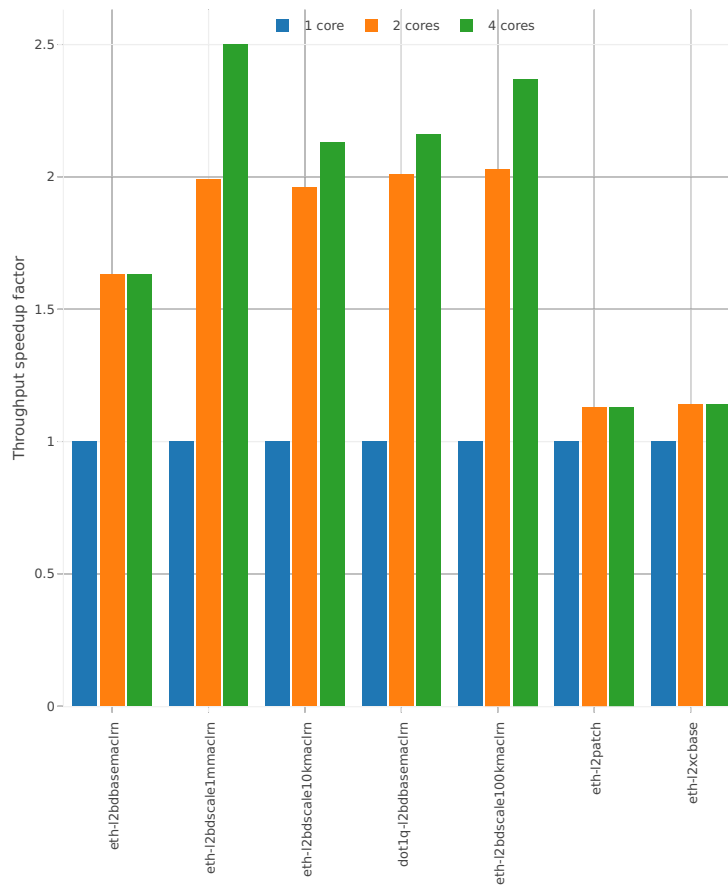
3n-skx-x710

64b-base_and_scale

Speedup: l2sw-3n-skx-x710-64b-base_and_scale-ndr

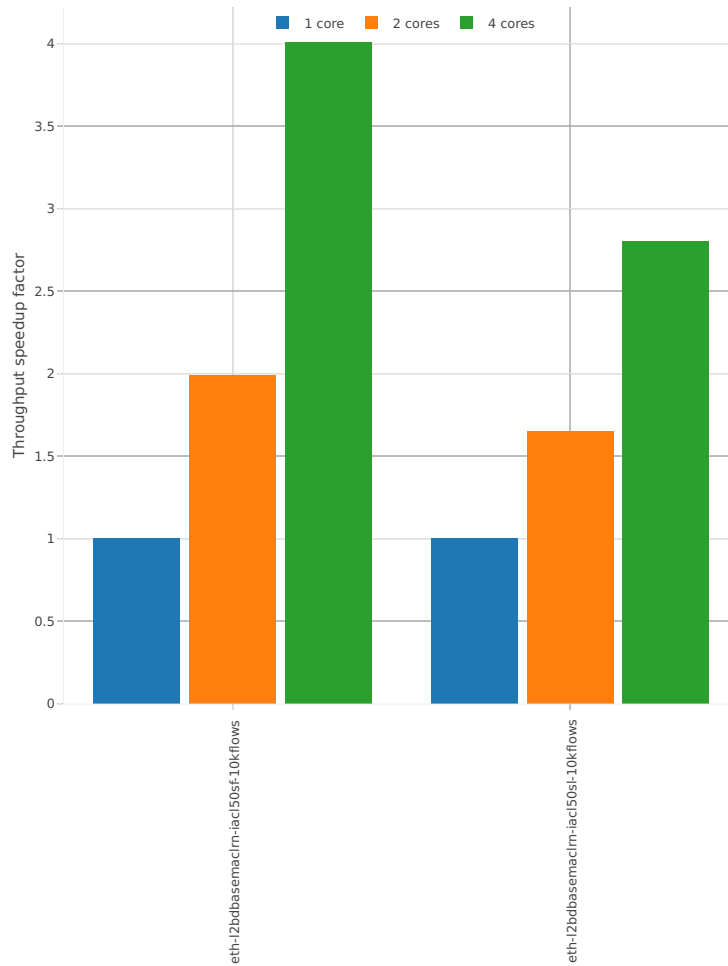


Speedup: l2sw-3n-skx-x710-64b-base_and_scale-pdr

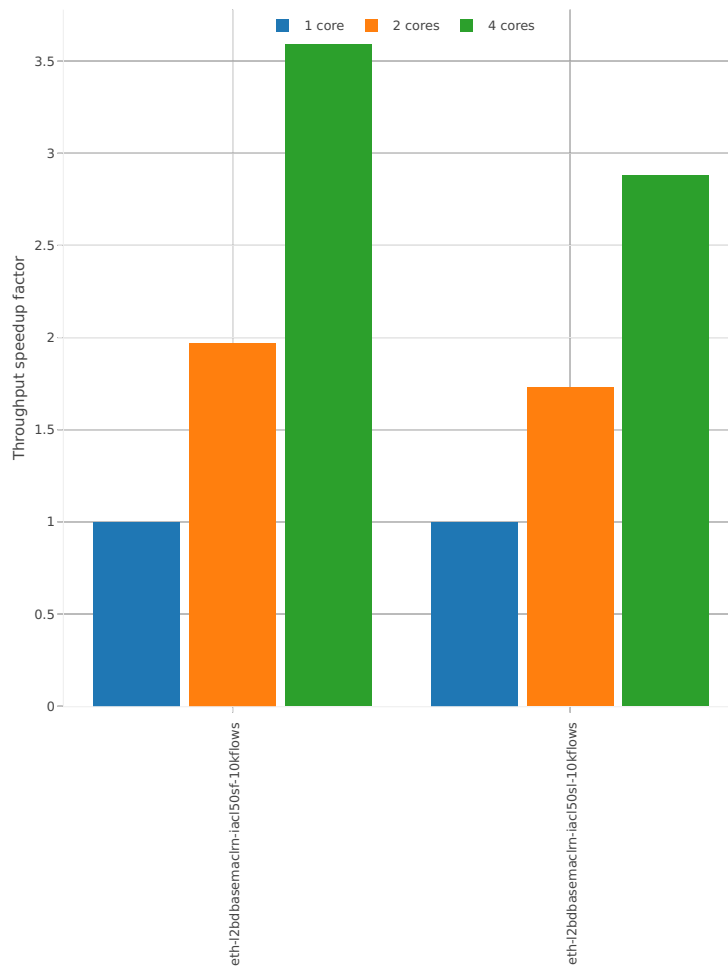


64b-features

Speedup: l2sw-3n-skx-x710-64b-features-ndr



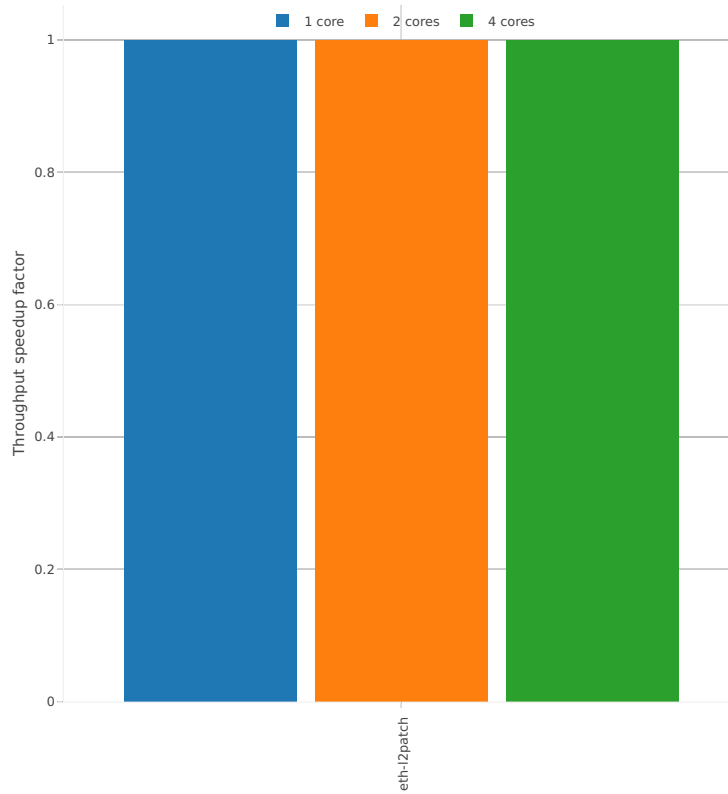
Speedup: l2sw-3n-skx-x710-64b-features-pdr



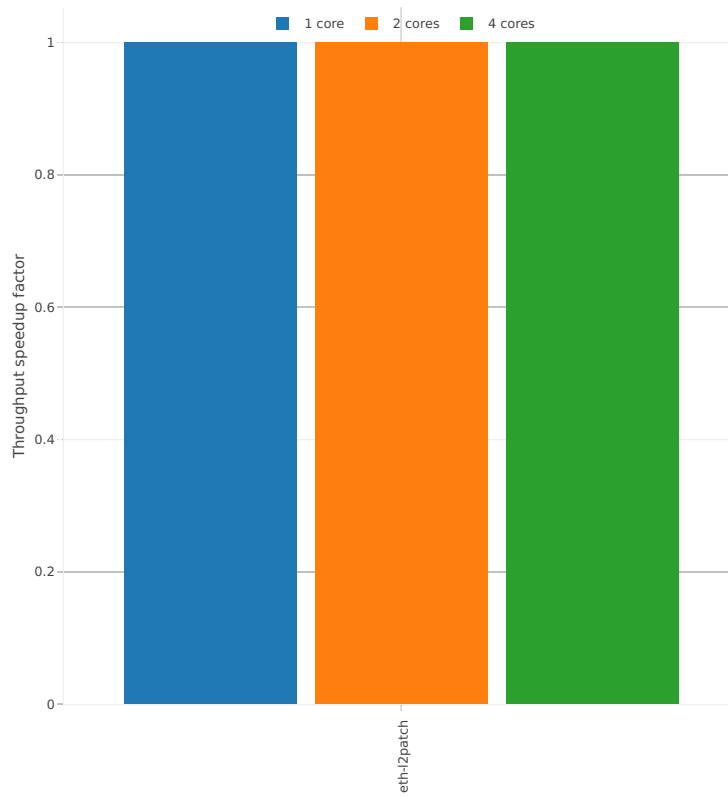
3n-skx-xxv710

64b-base_and_scale

Speedup: l2sw-3n-skx-xxv710-64b-base_and_scale-ndr



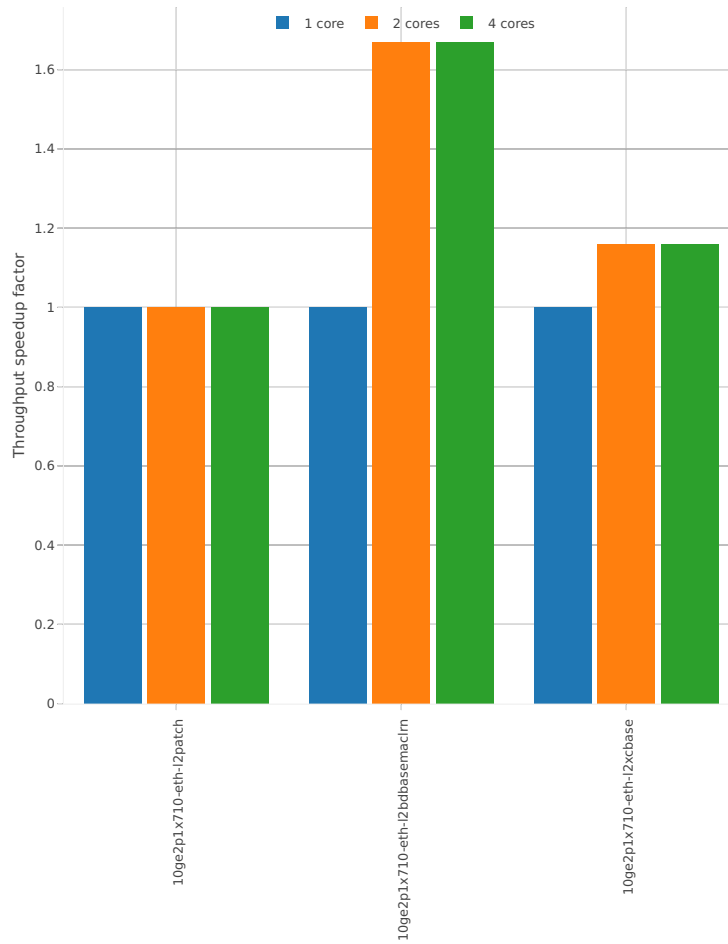
Speedup: l2sw-3n-skx-xxv710-64b-base_and_scale-pdr



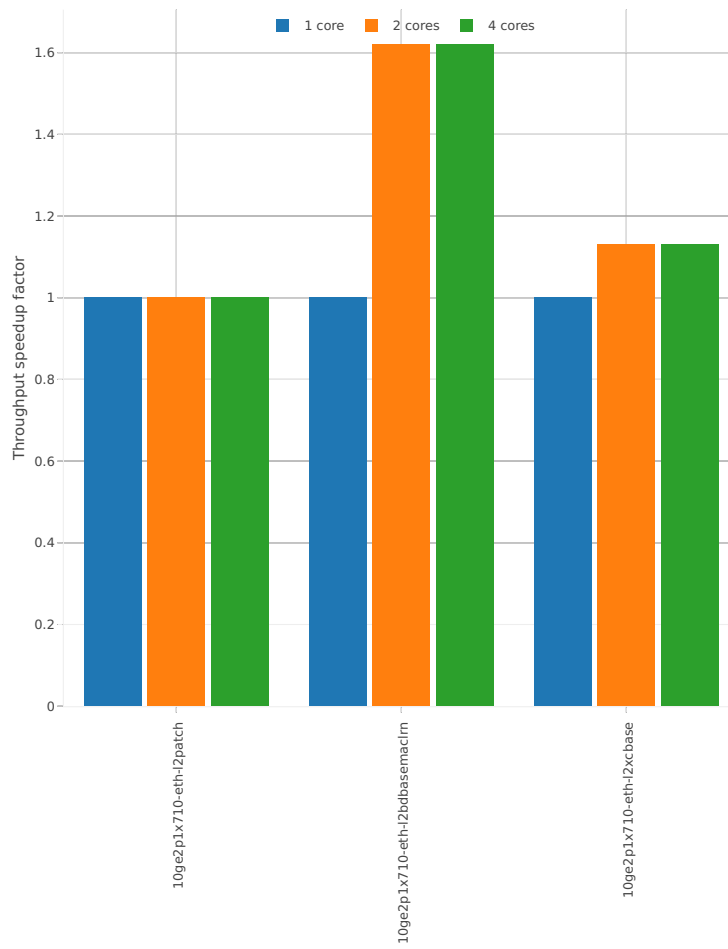
2n-skx-x710

64b-base_and_scale

Speedup: l2sw-2n-skx-x710-64b-base_and_scale-ndr



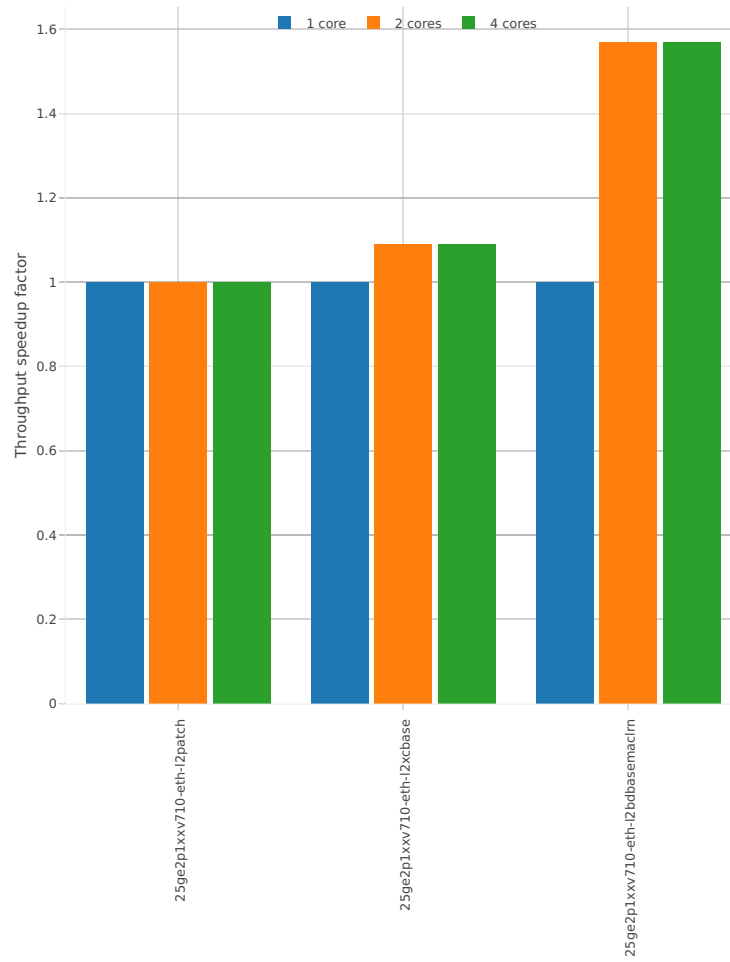
Speedup: l2sw-2n-skx-x710-64b-base_and_scale-pdr



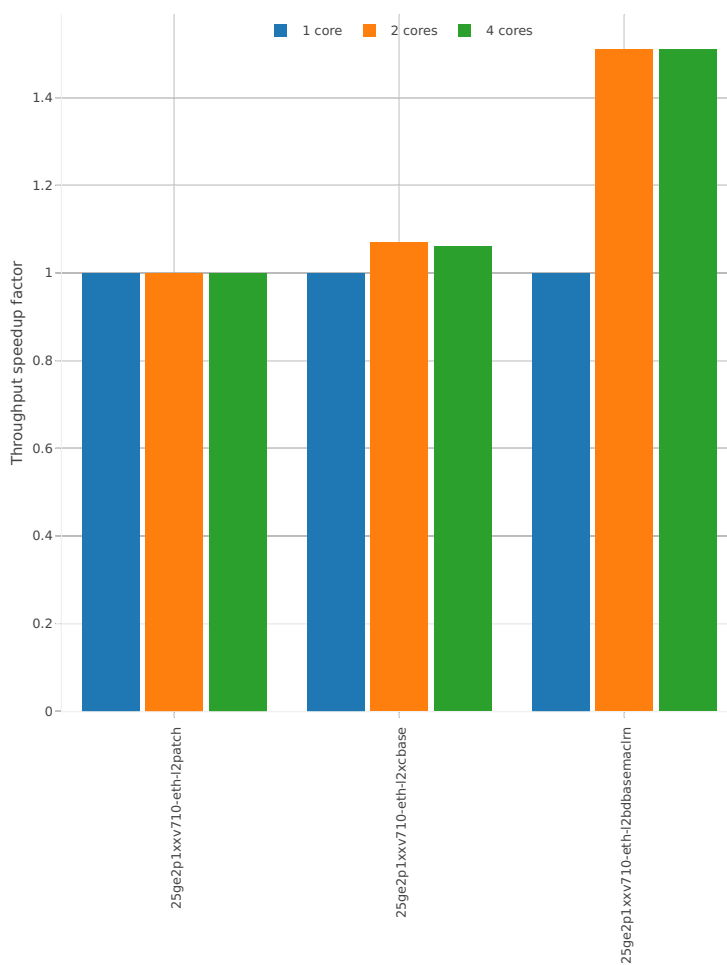
2n-skx-xxv710

64b-base_and_scale

Speedup: l2sw-2n-skx-xxv710-64b-base_and_scale-ndr



Speedup: l2sw-2n-skx-xxv710-64b-base_and_scale-pdr



2.4.2 IPv4 Routing

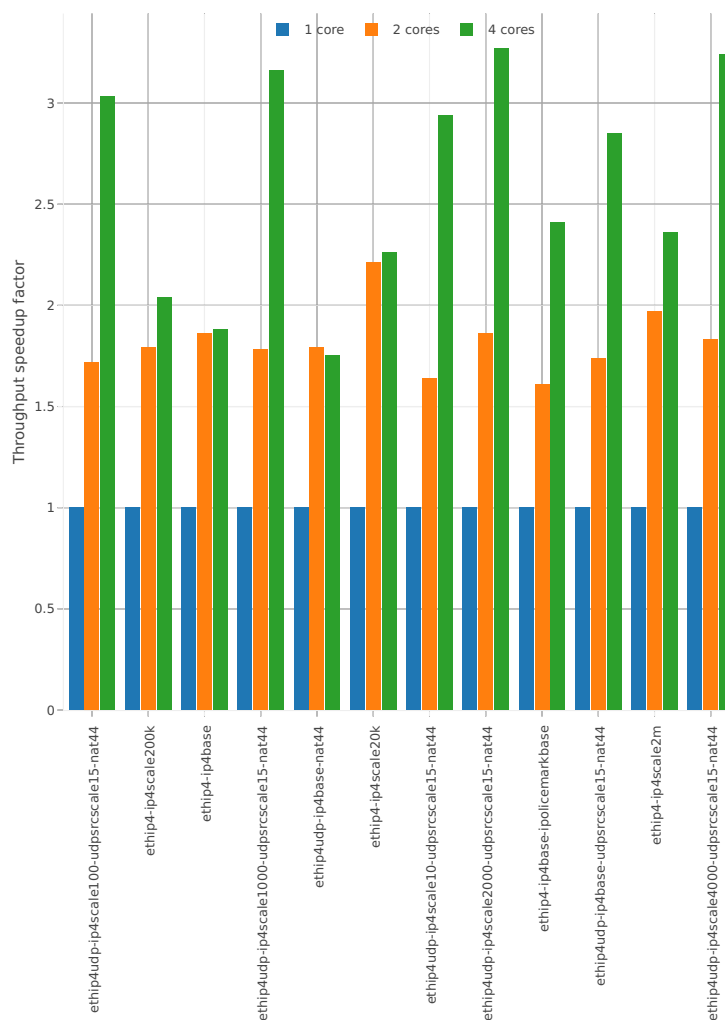
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 64B performance tests with VPP IPv4 Routed-Forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

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

3n-hsw-x520

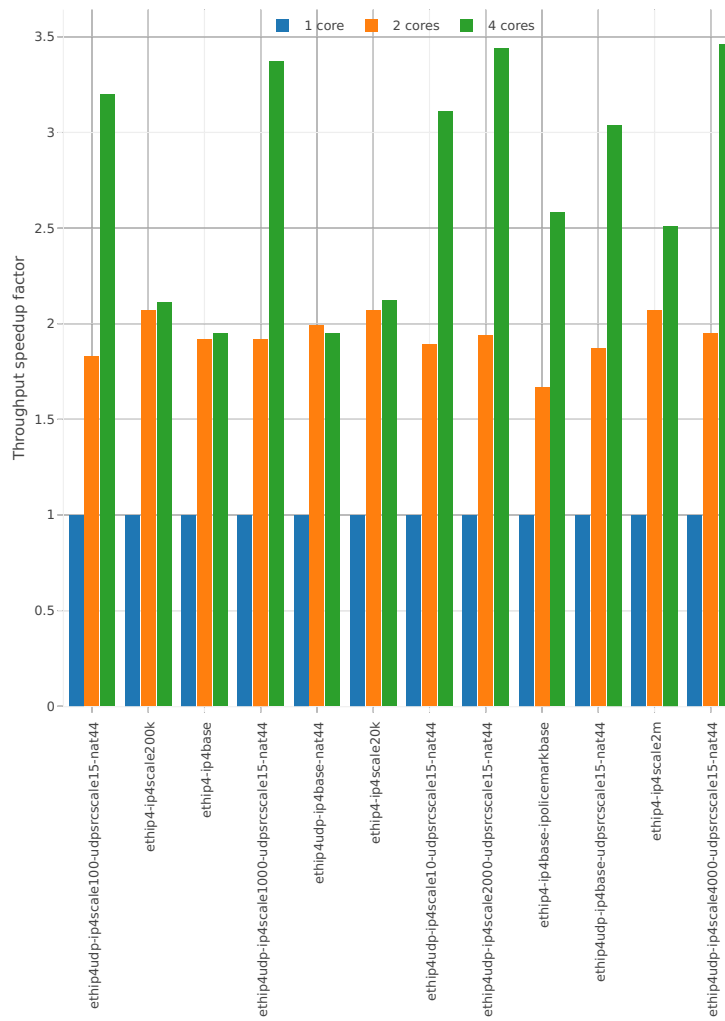
64b-base_and_scale

Speedup: ip4-3n-hsw-x520-64b-base_and_scale-ndr



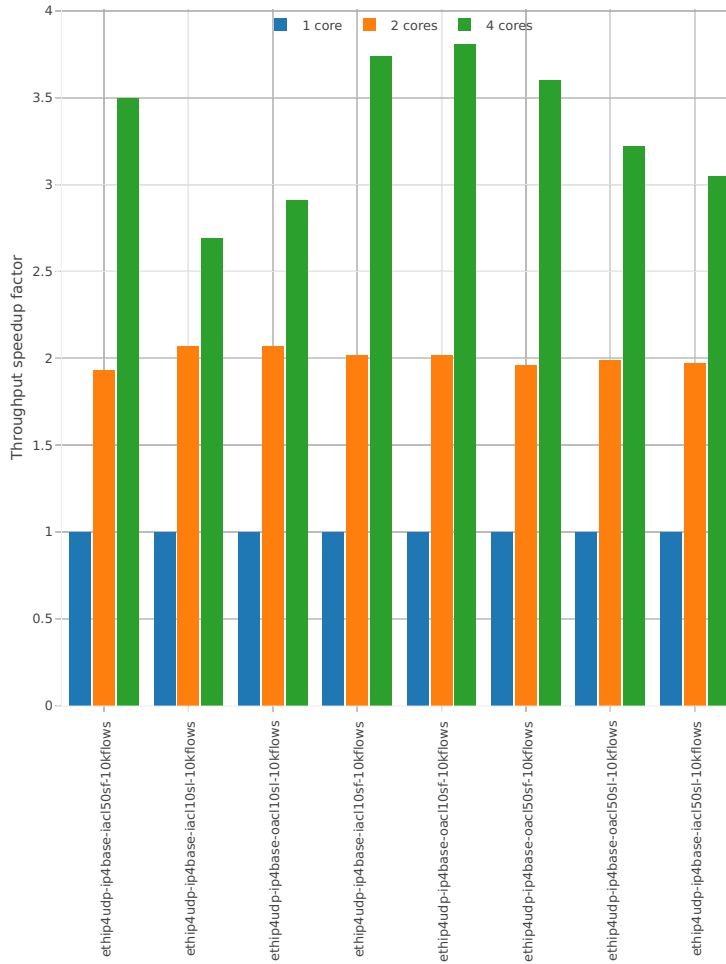
⁴⁵ <https://git.fd.io/csit/tree/tests/vpp/perf/ip4?h=rls1807>

Speedup: ip4-3n-hsw-x520-64b-base_and_scale-pdr

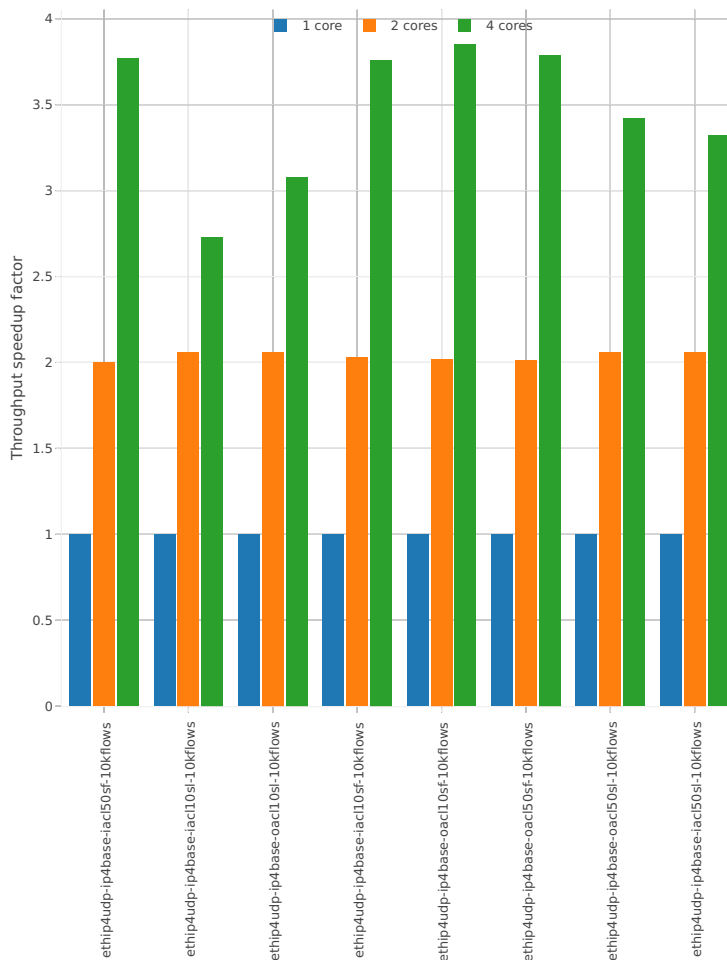


64b-features

Speedup: ip4-3n-hsw-x520-64b-features-ndr



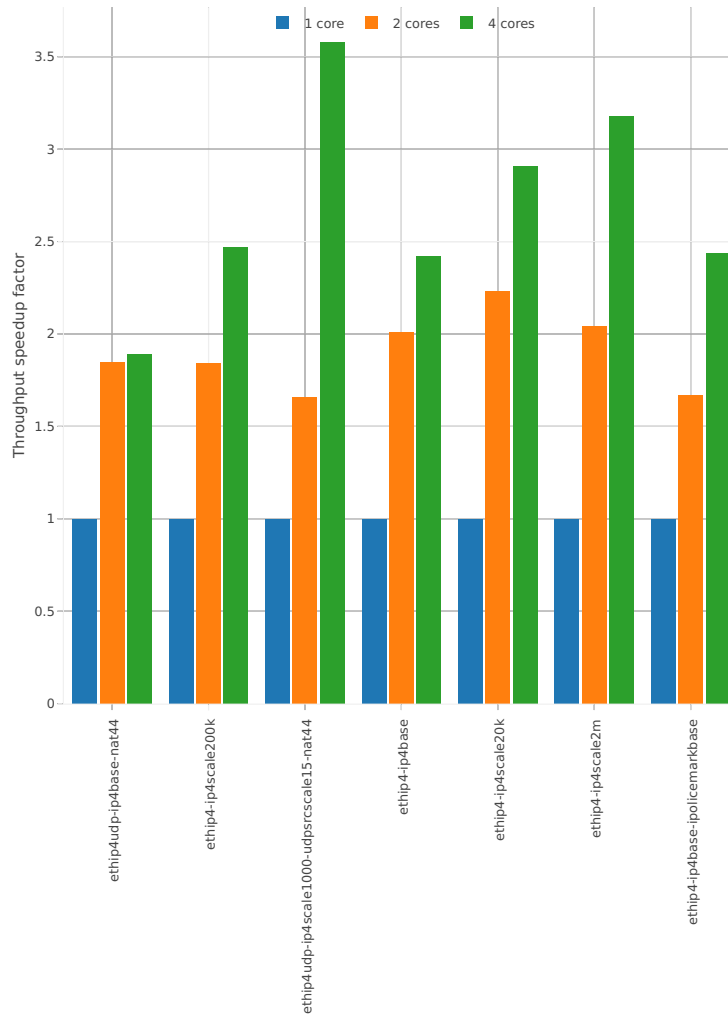
Speedup: ip4-3n-hsw-x520-64b-features-pdr



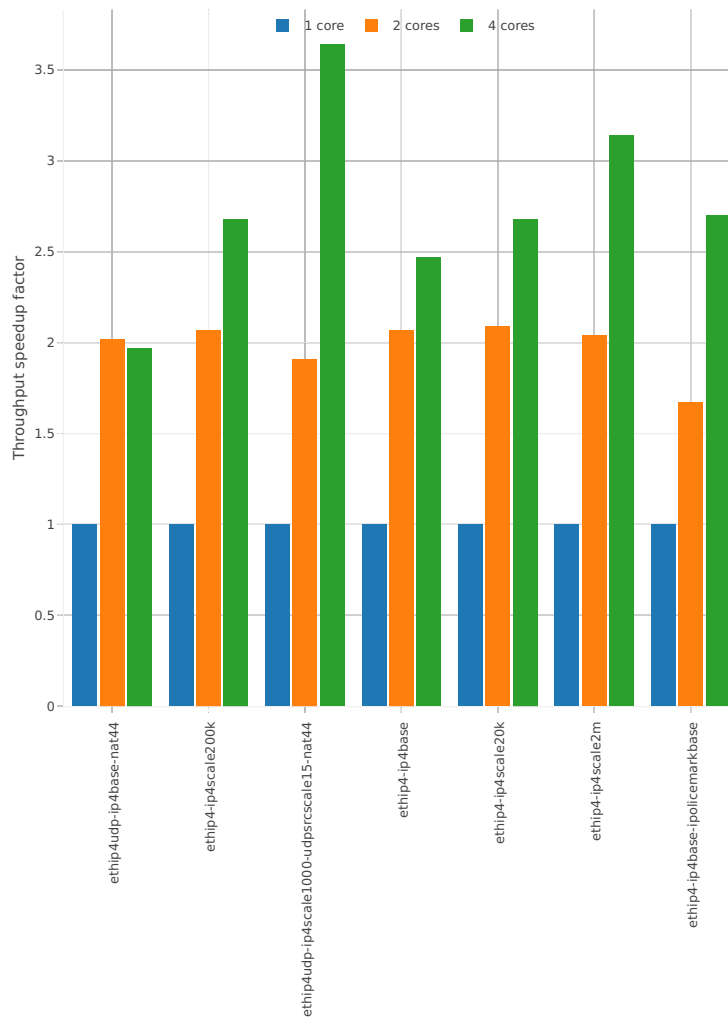
3n-hsw-x710

64b-base_and_scale

Speedup: ip4-3n-hsw-x710-64b-base_and_scale-ndr

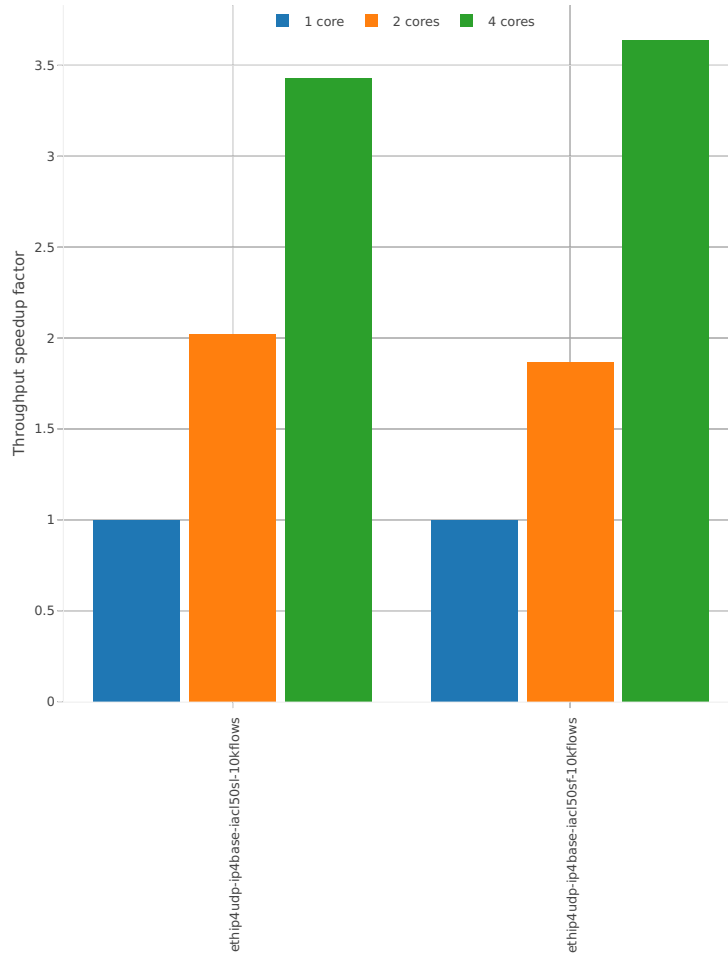


Speedup: ip4-3n-hsw-x710-64b-base_and_scale-pdr

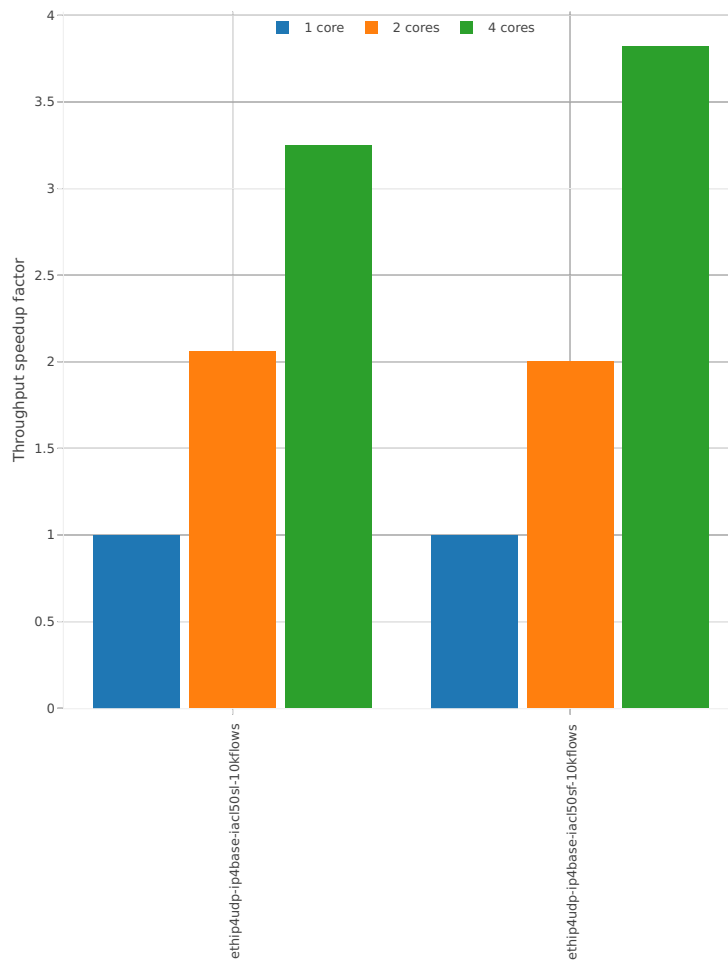


64b-features

Speedup: ip4-3n-hsw-x710-64b-features-ndr



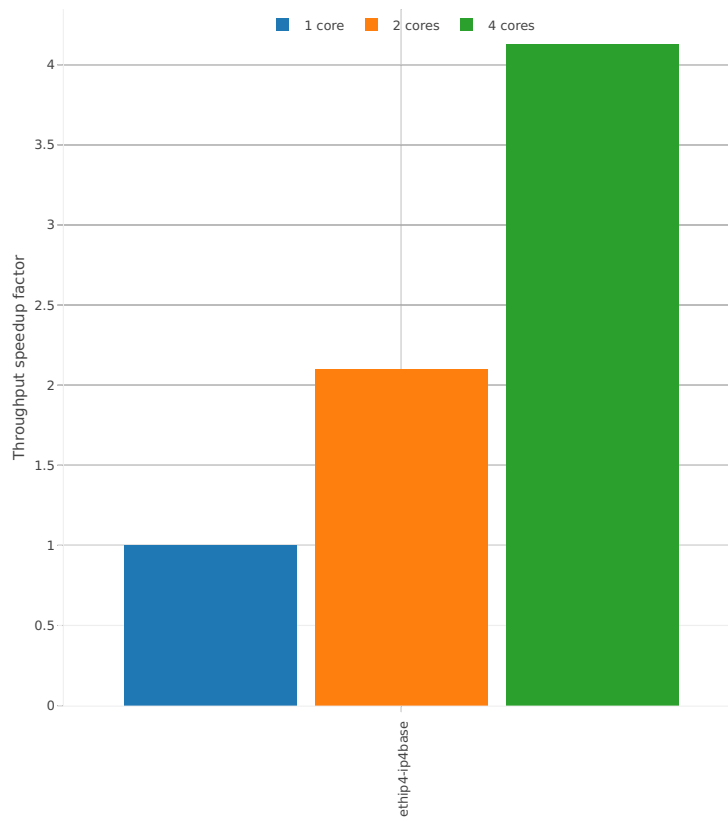
Speedup: ip4-3n-hsw-x710-64b-features-pdr



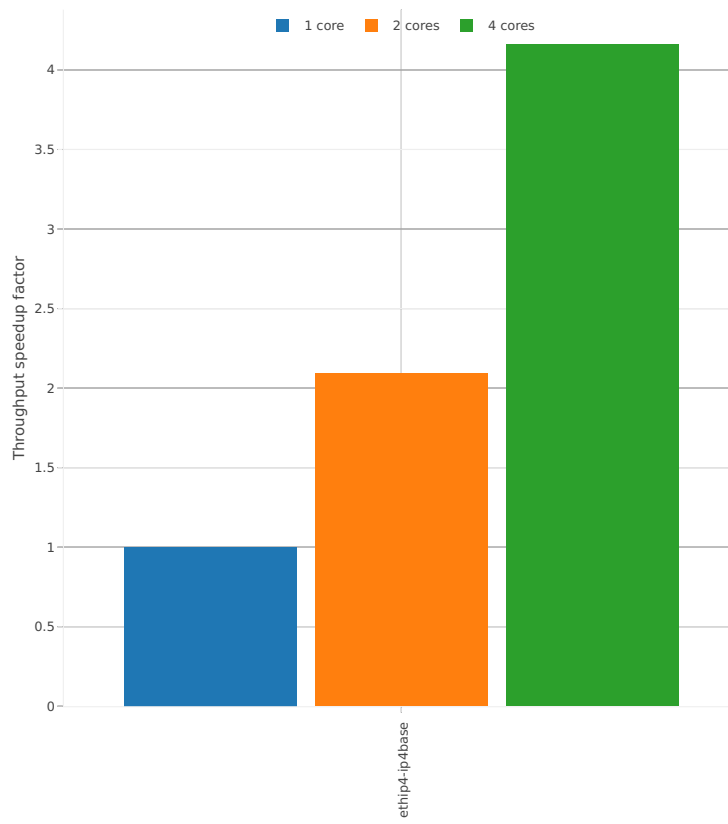
3n-hsw-xl710

64b-base_and_scale

Speedup: ip4-3n-hsw-xl710-64b-base_and_scale-ndr



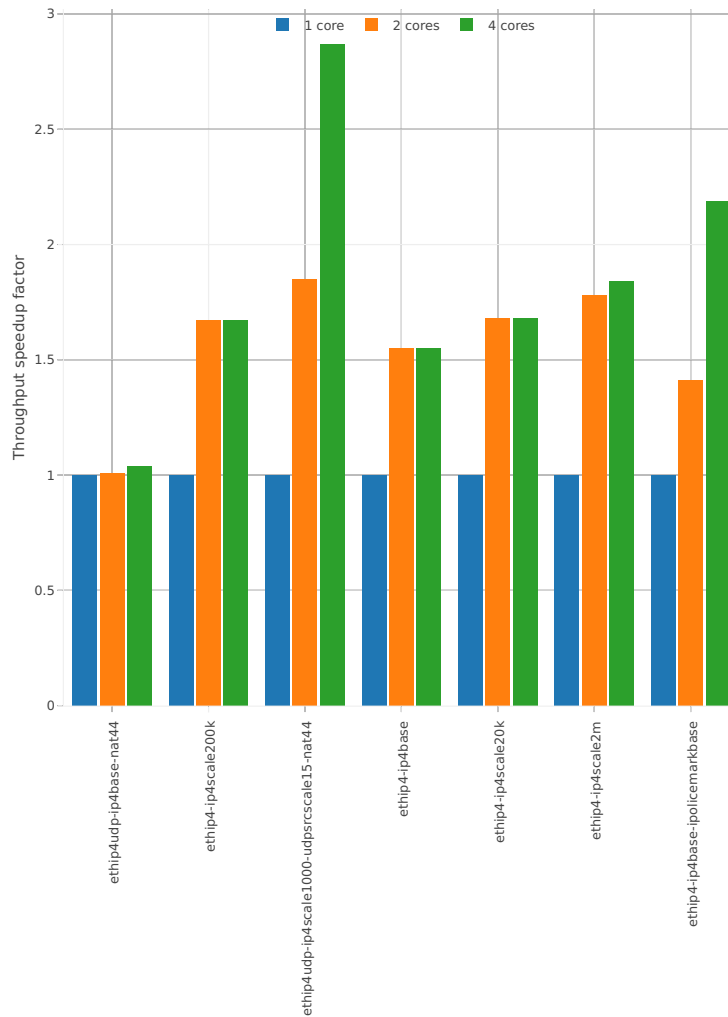
Speedup: ip4-3n-hsw-xl710-64b-base_and_scale-pdr



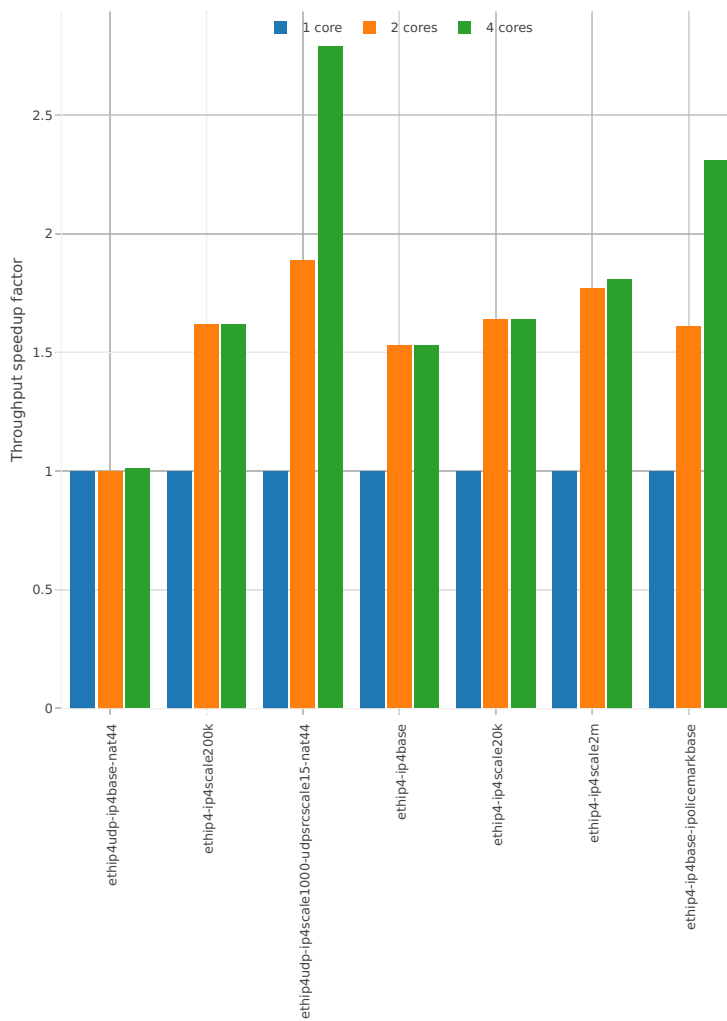
3n-skx-x710

64b-base_and_scale

Speedup: ip4-3n-skx-x710-64b-base_and_scale-ndr

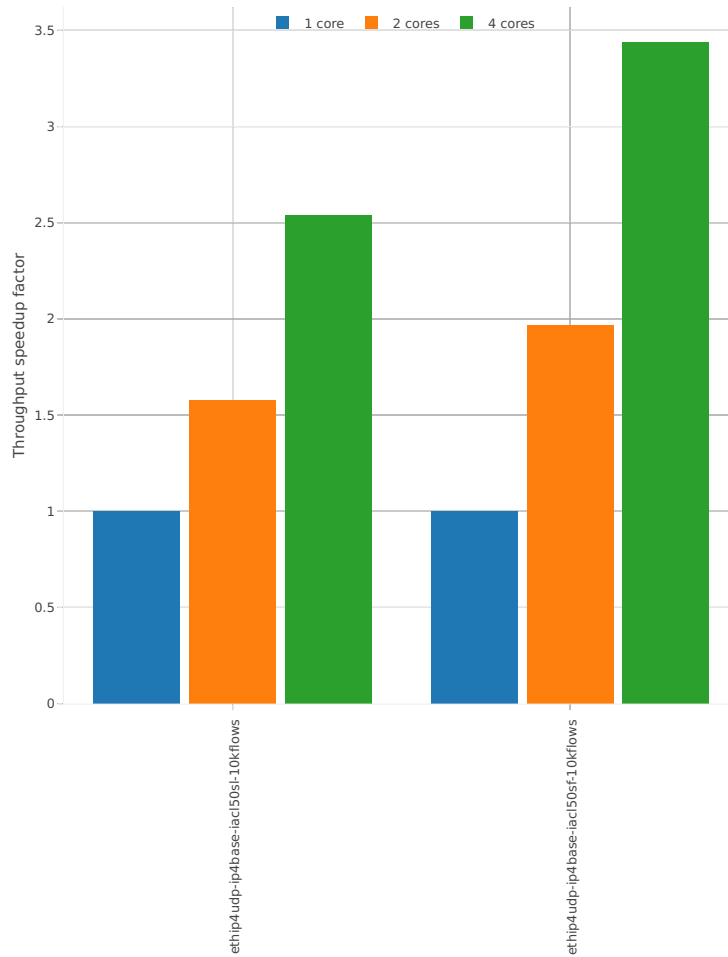


Speedup: ip4-3n-skx-x710-64b-base_and_scale-pdr

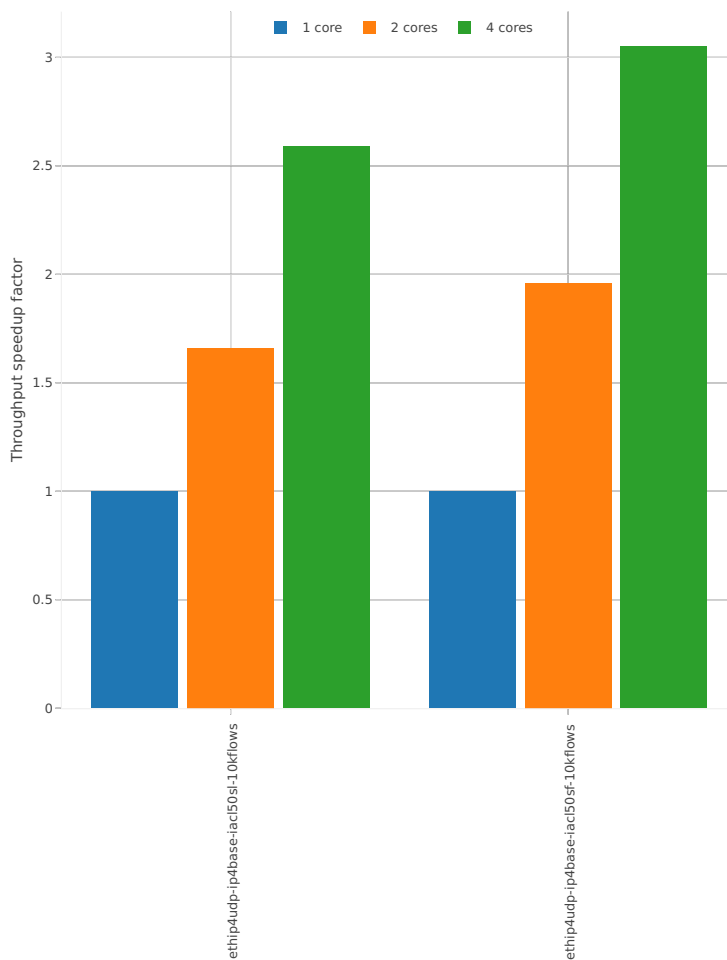


64b-features

Speedup: ip4-3n-skx-x710-64b-features-ndr

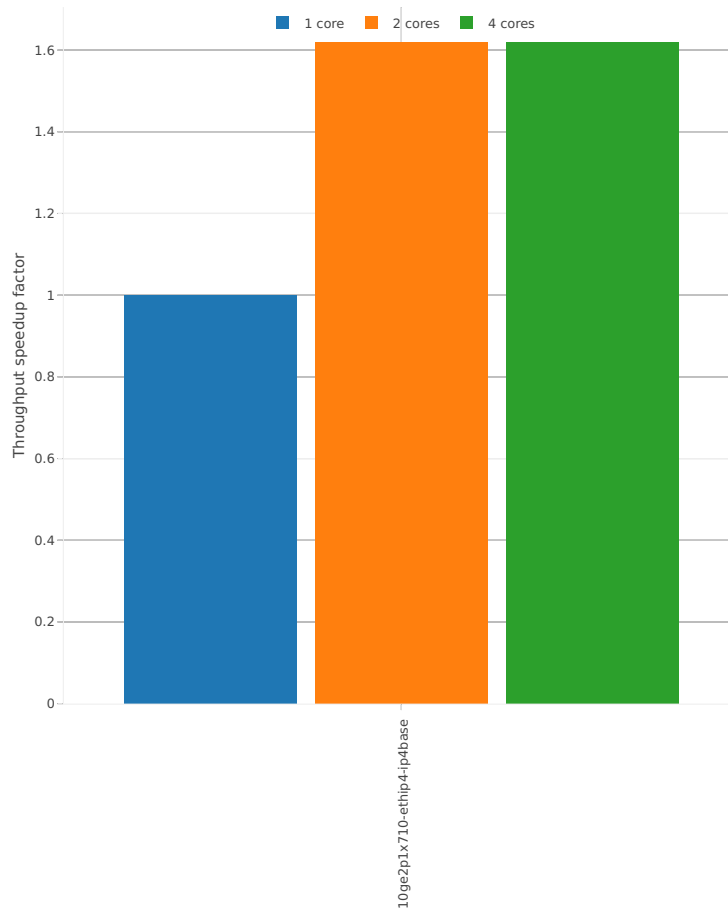


Speedup: ip4-3n-skx-x710-64b-features-pdr

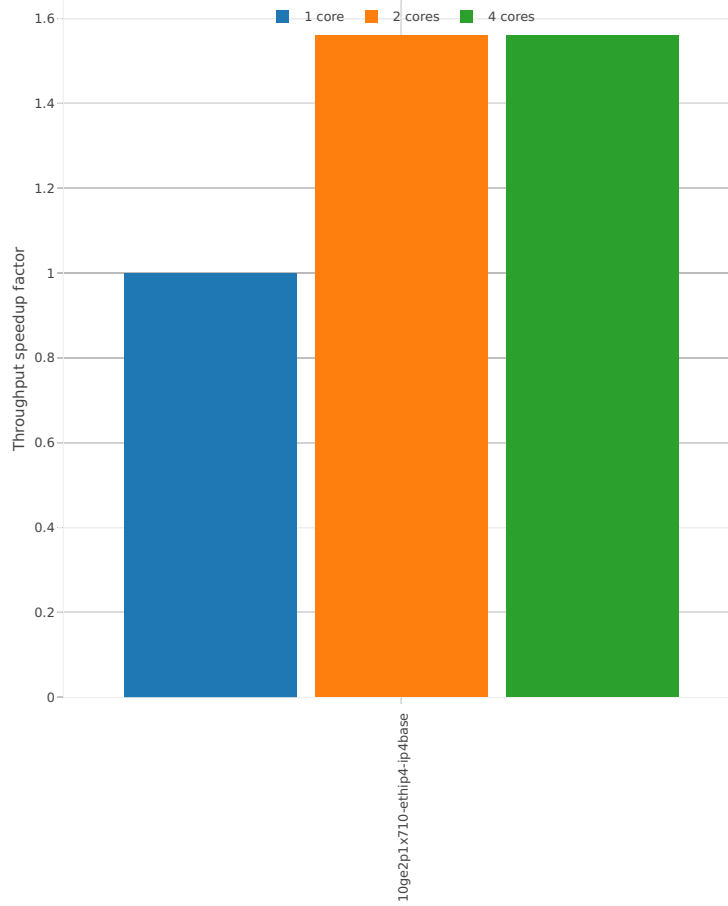


2n-skx-x710**64b-base_and_scale**

Speedup: ip4-2n-skx-x710-64b-base_and_scale-ndr



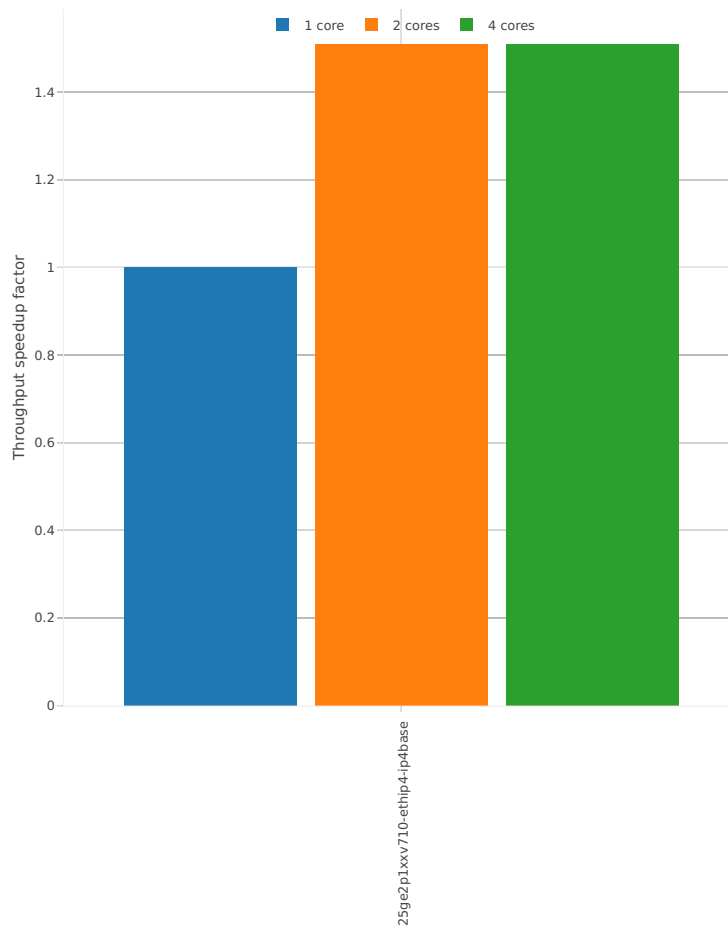
Speedup: ip4-2n-skx-x710-64b-base_and_scale-pdr



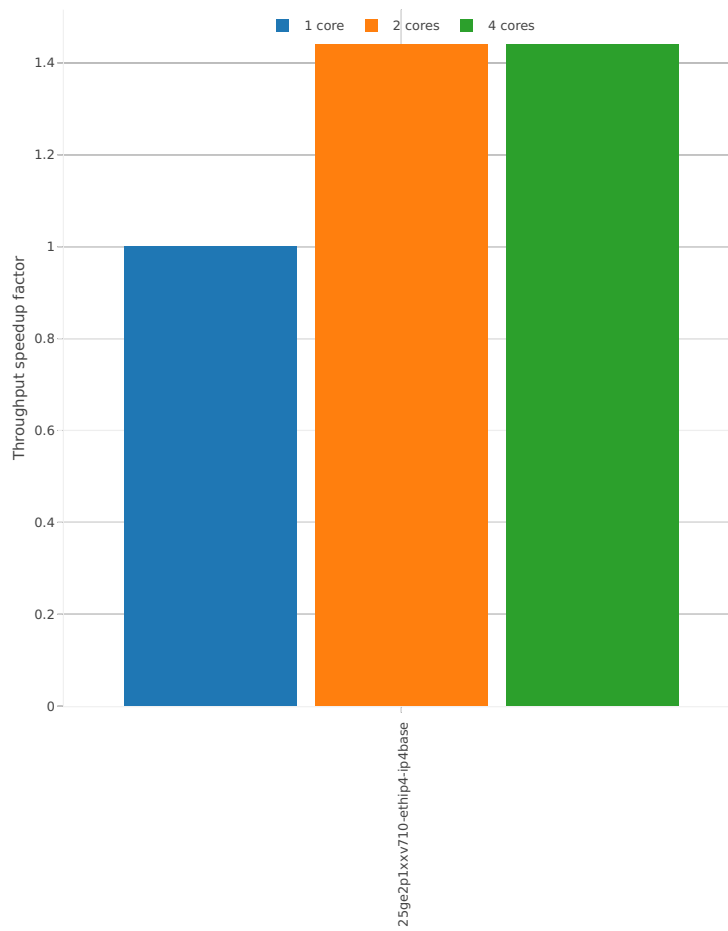
2n-skx-xxv710

64b-base_and_scale

Speedup: ip4-2n-skx-xxv710-64b-base_and_scale-ndr



Speedup: ip4-2n-skx-xxv710-64b-base_and_scale-pdr



2.4.3 IPv6 Routing

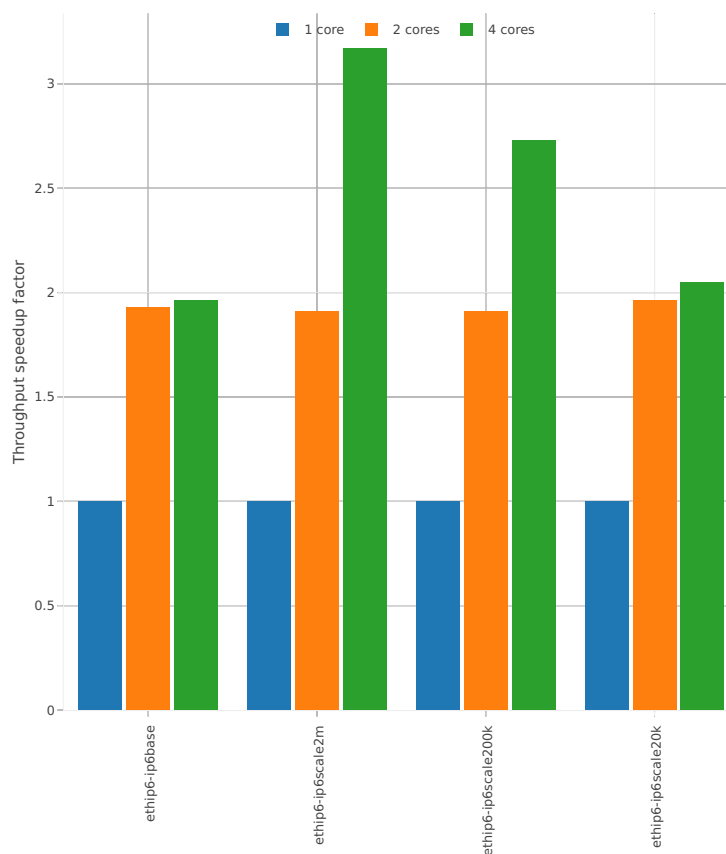
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 78B performance tests with VPP IPv6 Routed-Forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

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

3n-hsw-x520

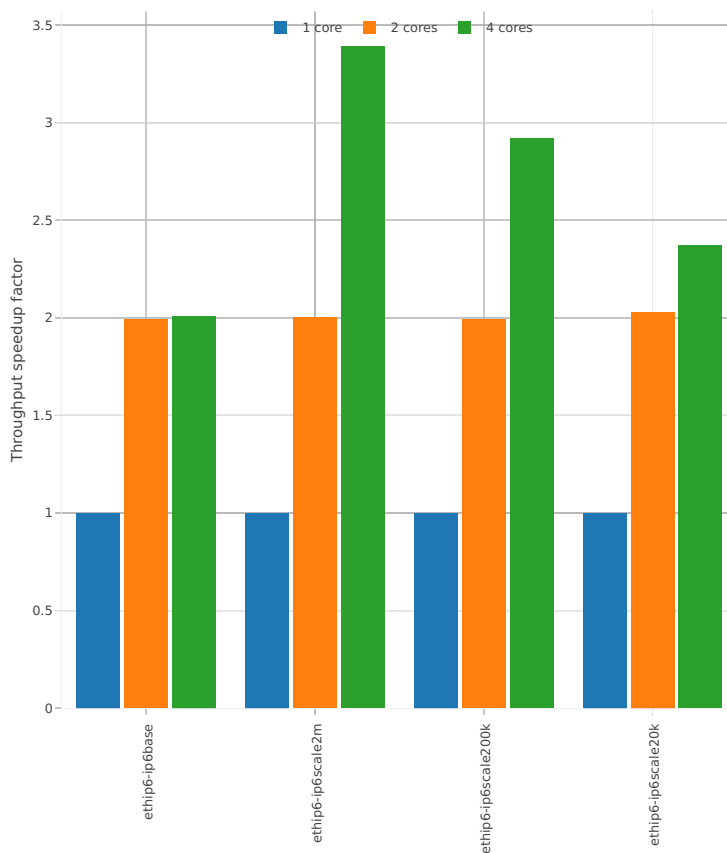
78b-base_and_scale

Speedup: ip6-3n-hsw-x520-78b-base_and_scale-ndr



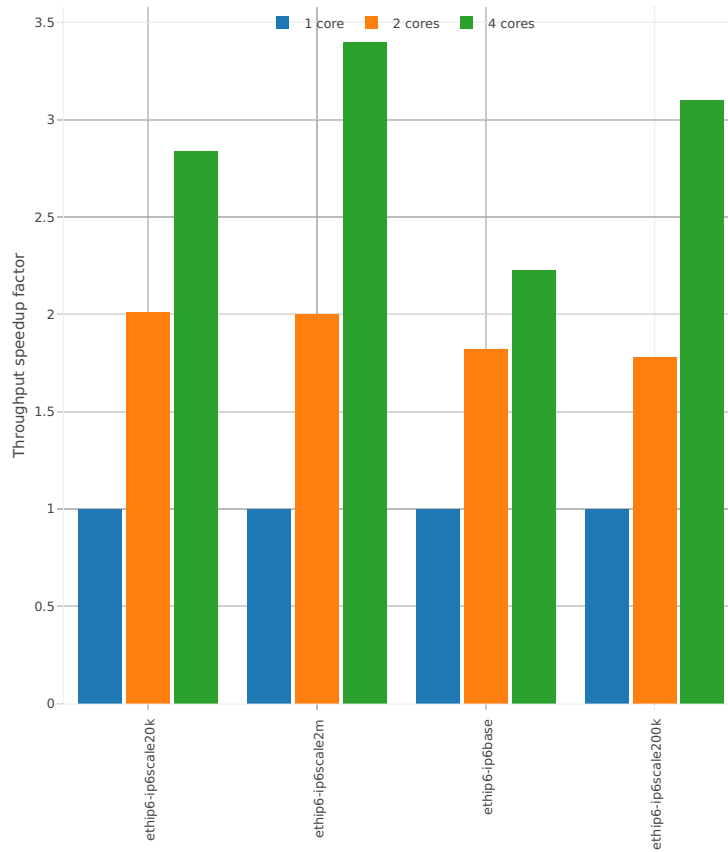
⁴⁶ <https://git.fd.io/csit/tree/tests/vpp/perf/ip6?h=rls1807>

Speedup: ip6-3n-hsw-x520-78b-base_and_scale-pdr

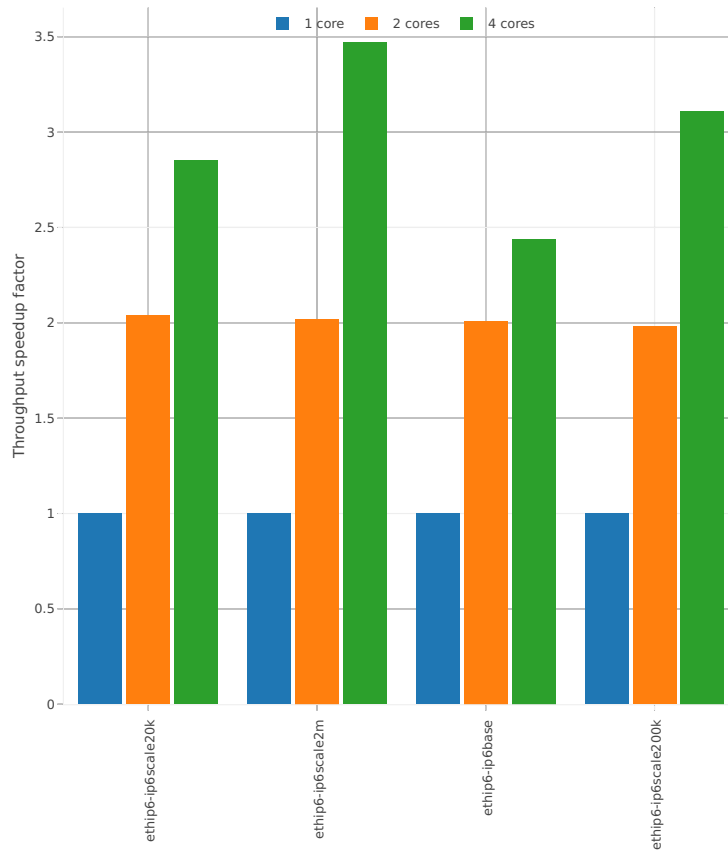


3n-hsw-x710**78b-base_and_scale**

Speedup: ip6-3n-hsw-x710-78b-base_and_scale-ndr



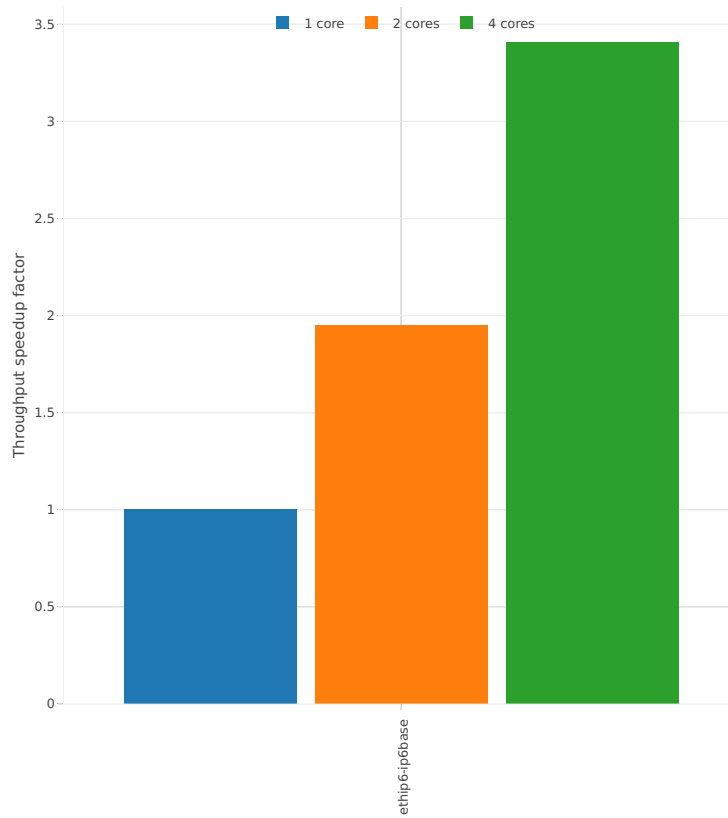
Speedup: ip6-3n-hsw-x710-78b-base_and_scale-pdr



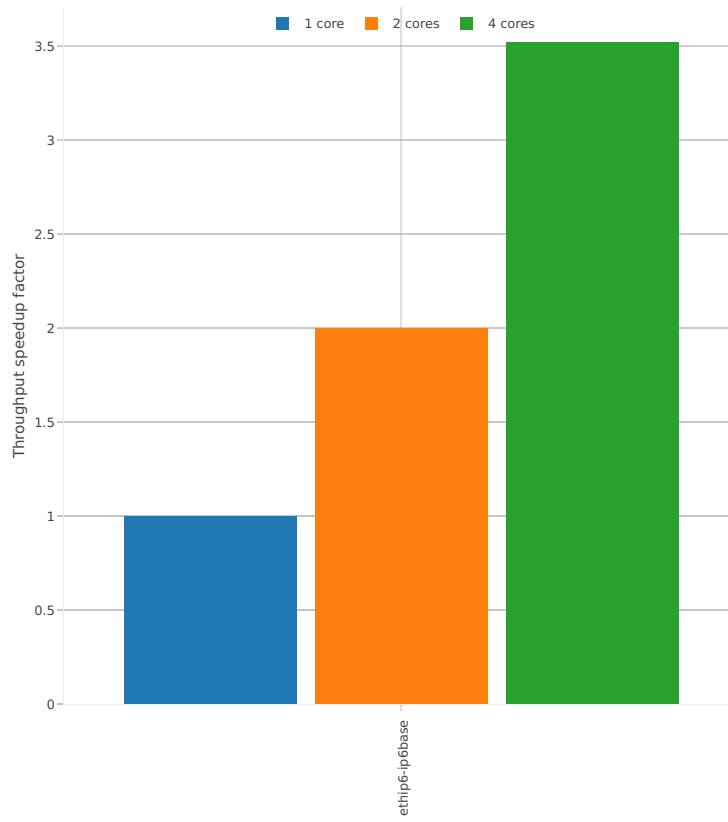
3n-hsw-xl710

78b-base_and_scale

Speedup: ip6-3n-hsw-xl710-78b-base_and_scale-ndr

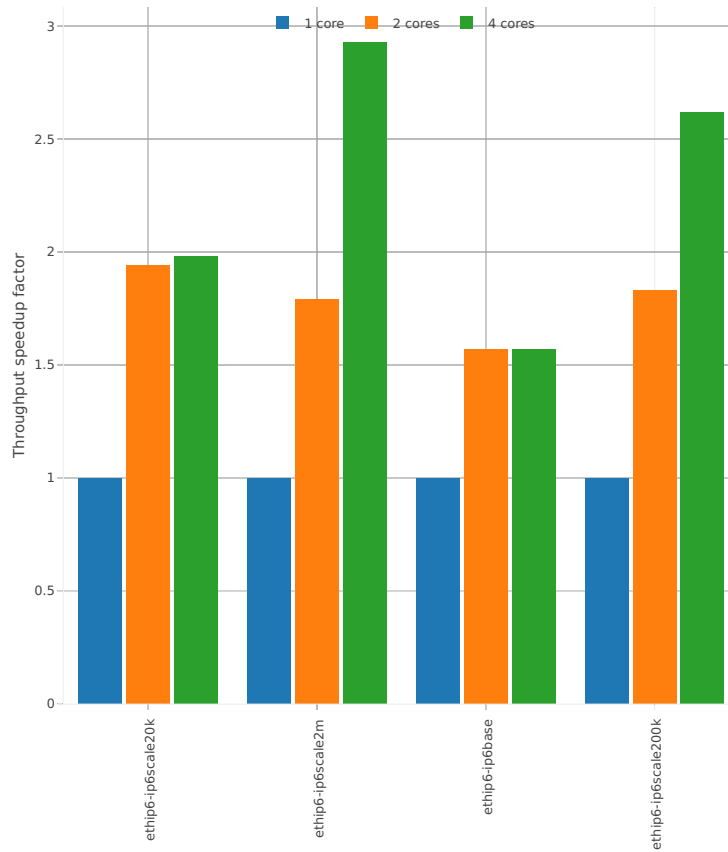


Speedup: ip6-3n-hsw-xl710-78b-base_and_scale-pdr

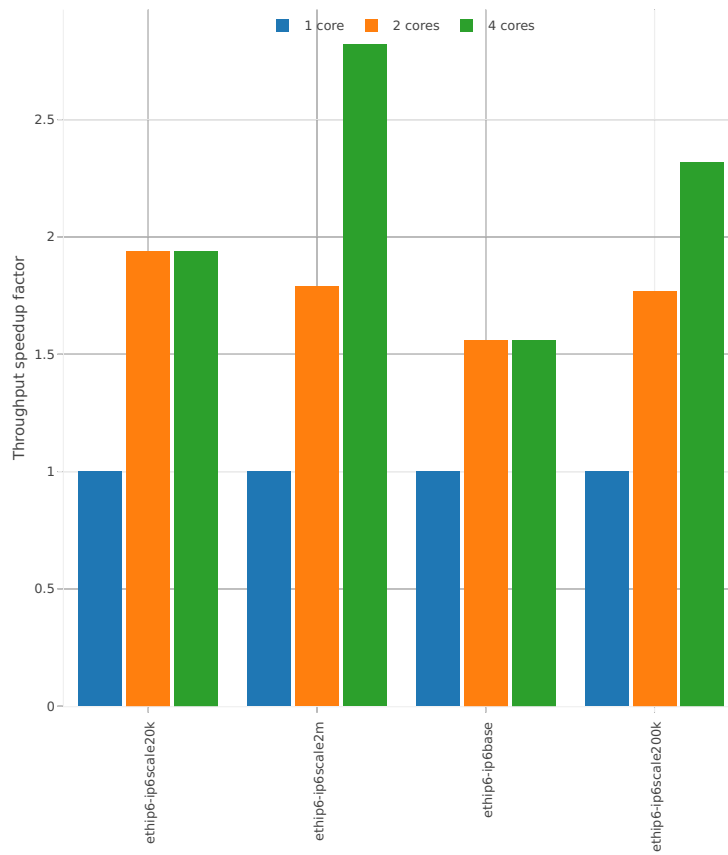


3n-skx-x710**78b-base_and_scale**

Speedup: ip6-3n-skx-x710-78b-base_and_scale-ndr



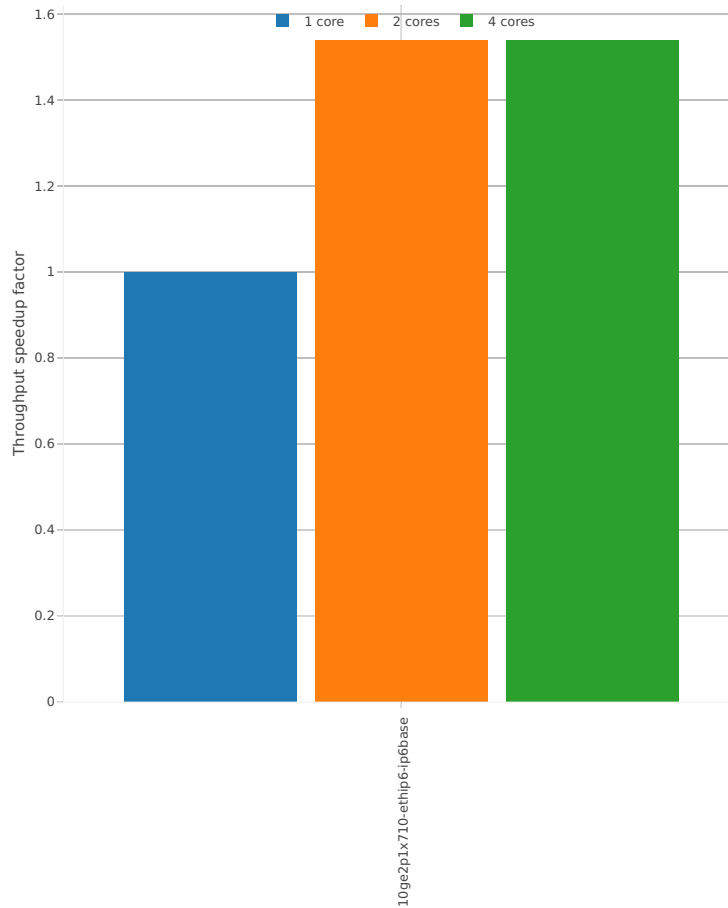
Speedup: ip6-3n-skx-x710-78b-base_and_scale-pdr



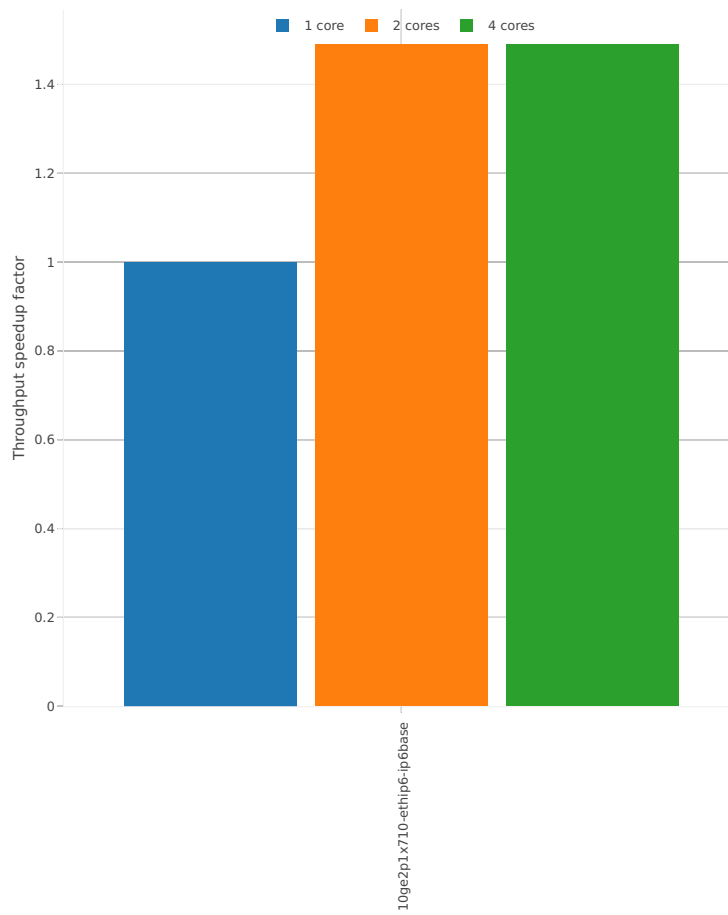
2n-skx-x710

78b-base_and_scale

Speedup: ip6-2n-skx-x710-78b-base_and_scale-ndr



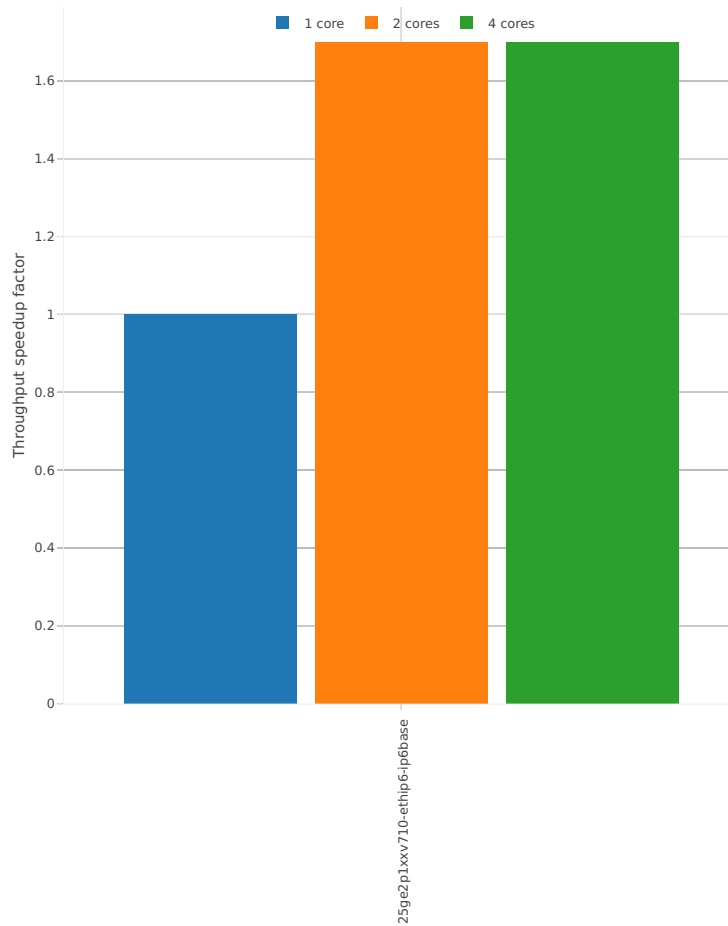
Speedup: ip6-2n-skx-x710-78b-base_and_scale-pdr



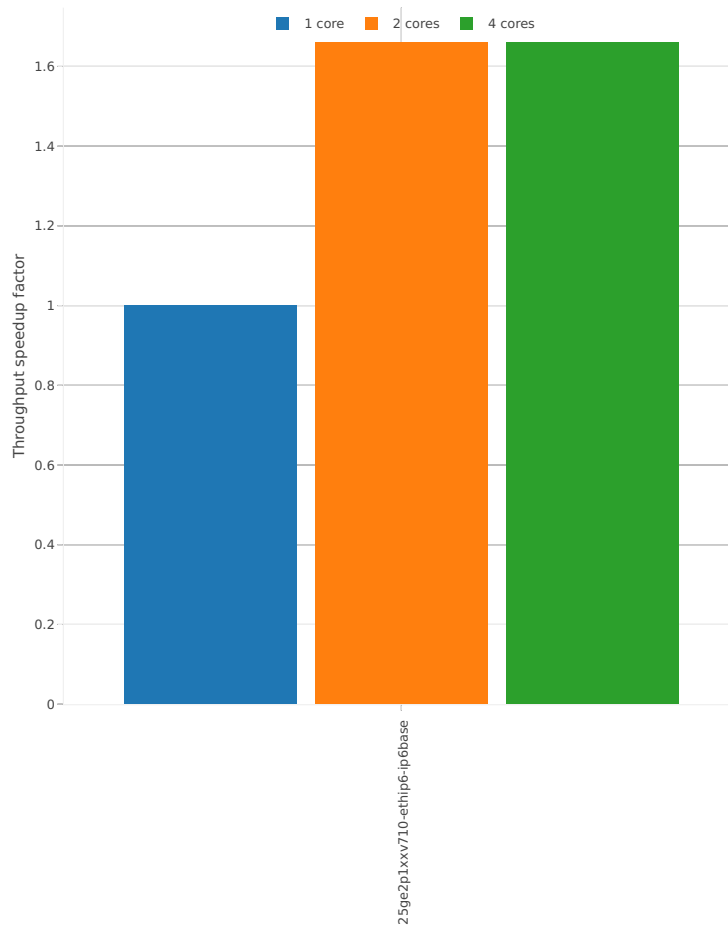
2n-skx-xxv710

78b-base_and_scale

Speedup: ip6-2n-skx-xxv710-78b-base_and_scale-ndr



Speedup: ip6-2n-skx-xxv710-78b-base_and_scale-pdr



2.4.4 SRv6 Routing

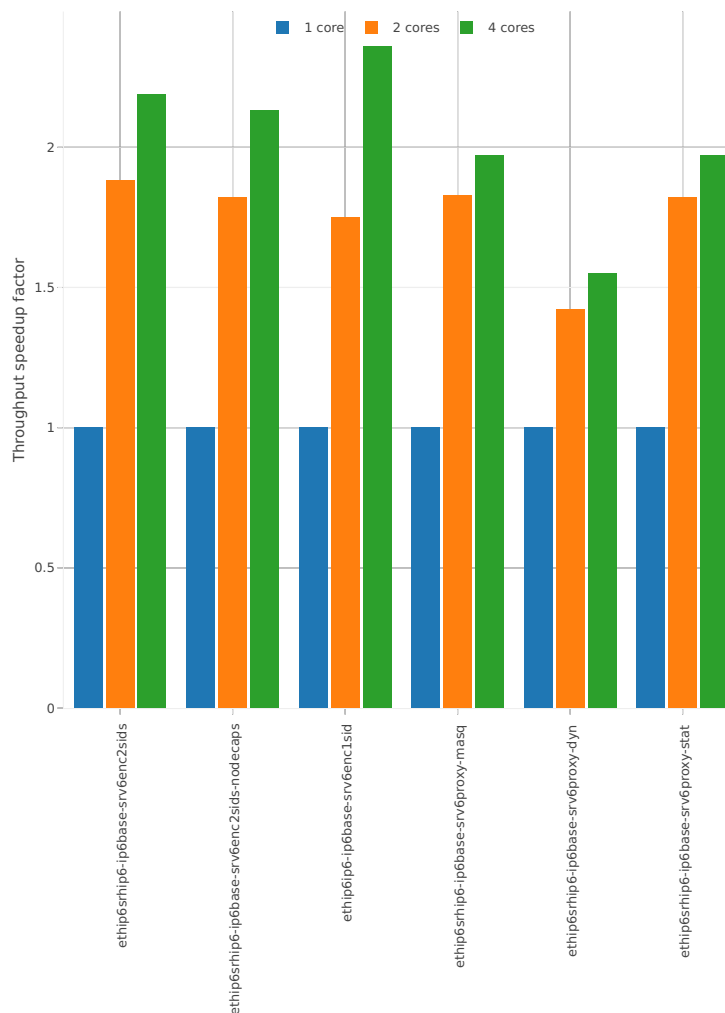
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 78B performance tests with VPP SRv6, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁴⁷.

3n-hsw-x520

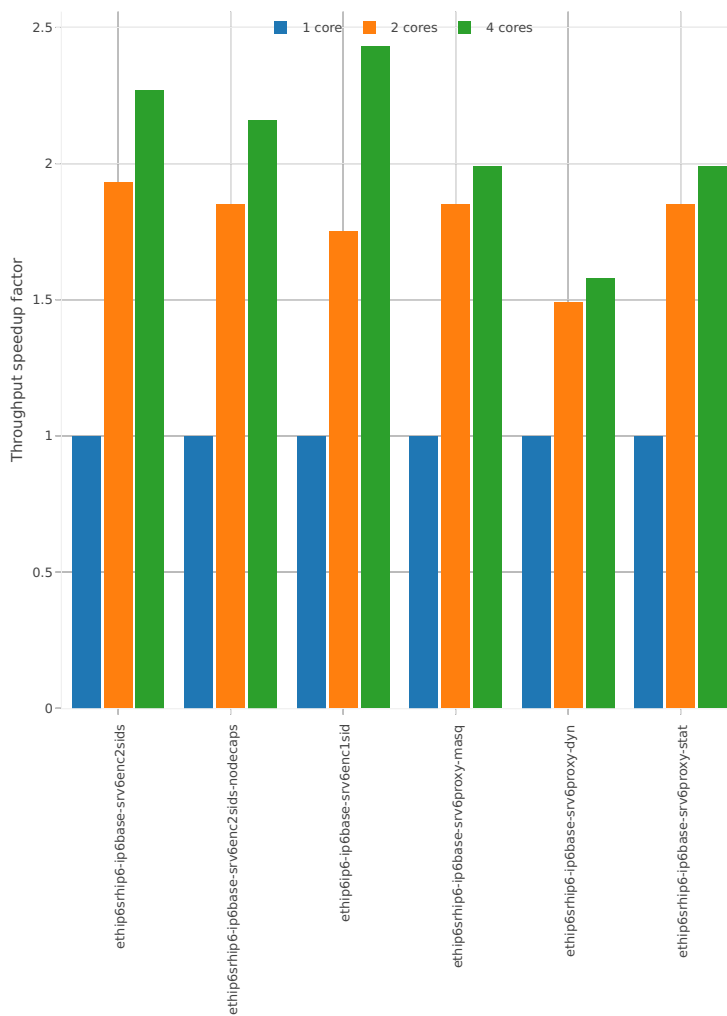
78b-features

Speedup: srv6-3n-hsw-x520-78b-1t1c-features-ndr



⁴⁷ <https://git.fd.io/csit/tree/tests/vpp/perf/srv6?h=rls1807>

Speedup: srv6-3n-hsw-x520-78b-1t1c-features-pdr



2.4.5 IPv4 Tunnels

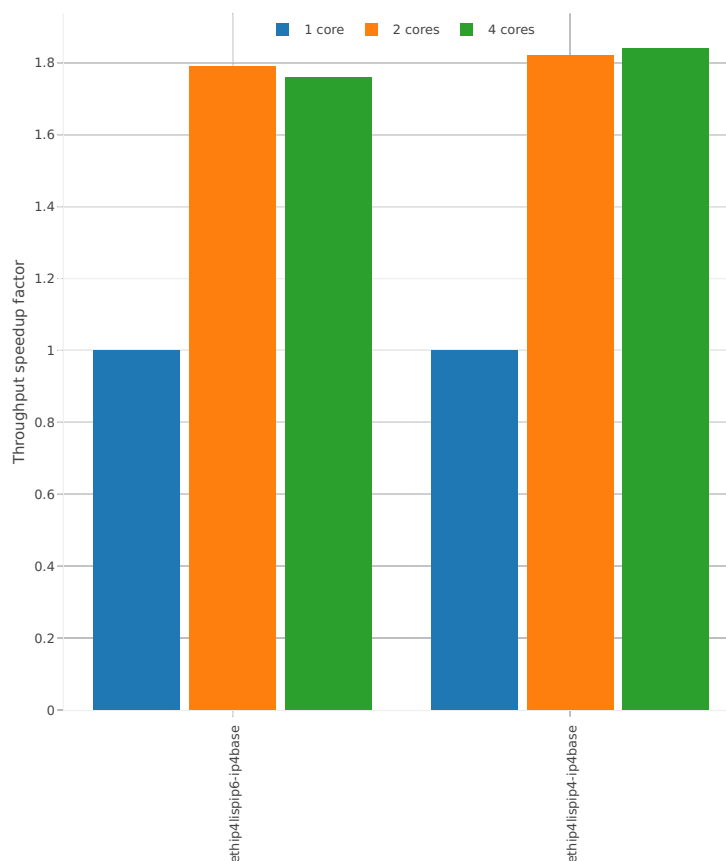
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Performance is reported for VPP 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](#)⁴⁸.

3n-hsw-x520

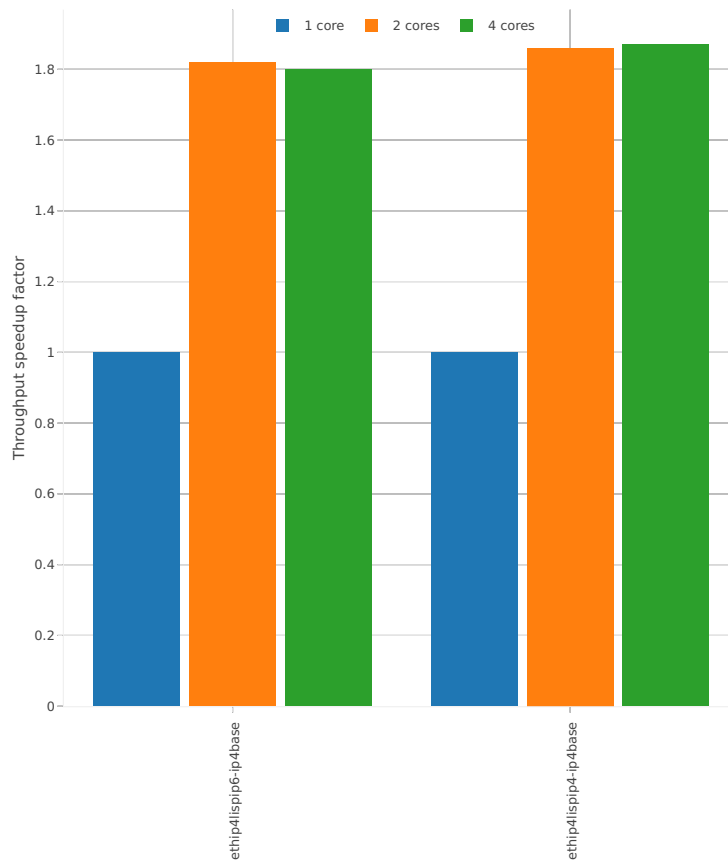
64b-base_and_scale

Speedup: ip4tun-3n-hsw-x520-64b-base_and_scale-ndr



⁴⁸ https://git.fd.io/csit/tree/tests/vpp/perf/ip4_tunnels?h=rls1807

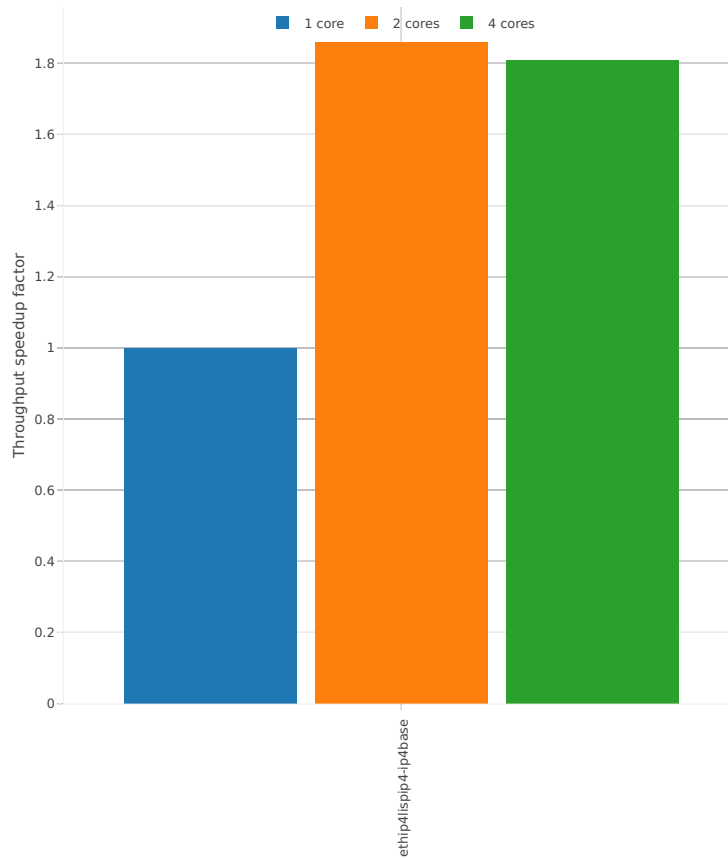
Speedup: ip4tun-3n-hsw-x520-64b-base_and_scale-pdr



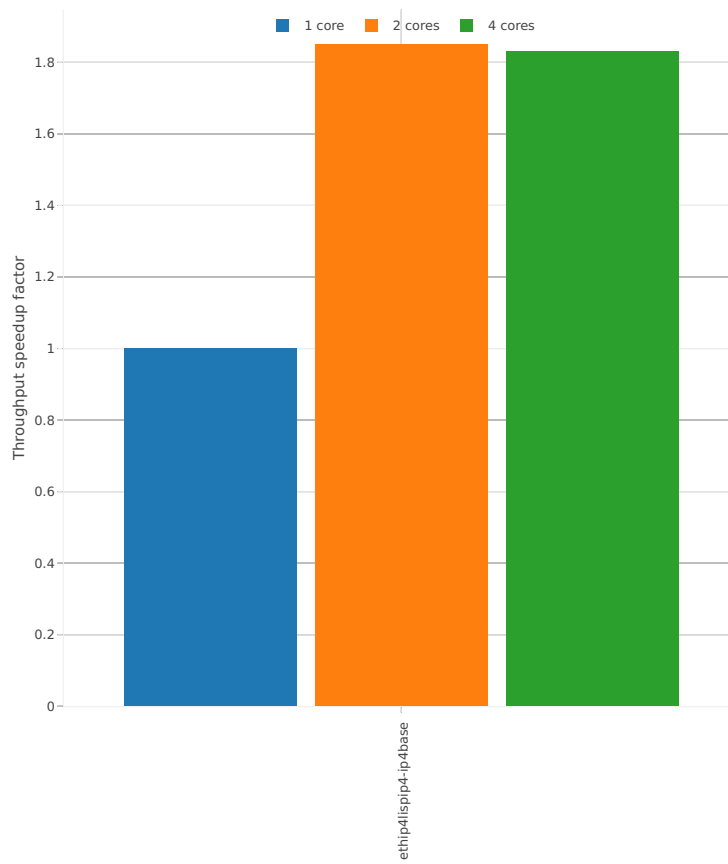
3n-hsw-x710

64b-base_and_scale

Speedup: ip4tun-3n-hsw-x710-64b-base_and_scale-ndr



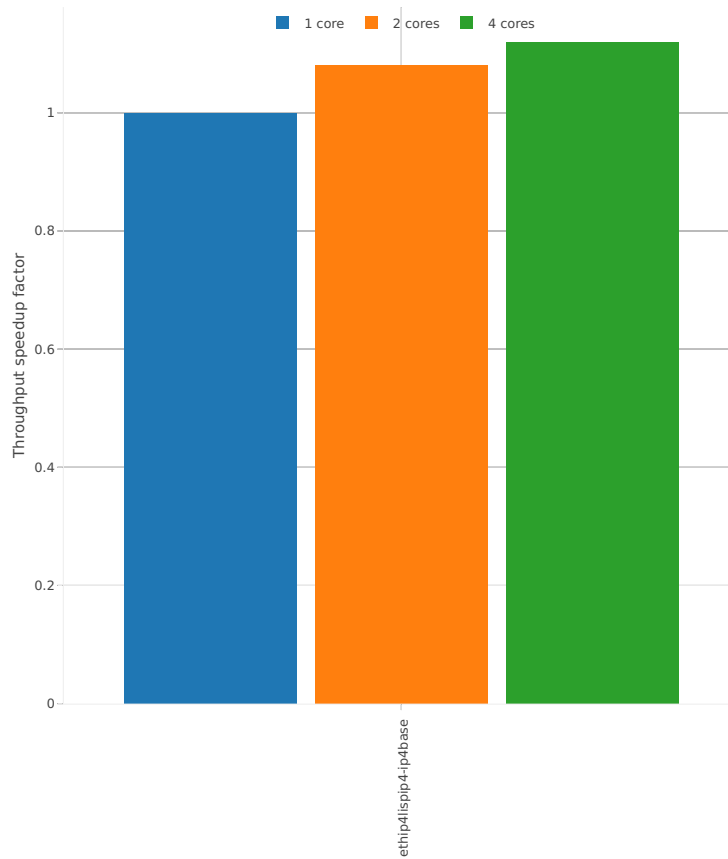
Speedup: ip4tun-3n-hsw-x710-64b-base_and_scale-pdr



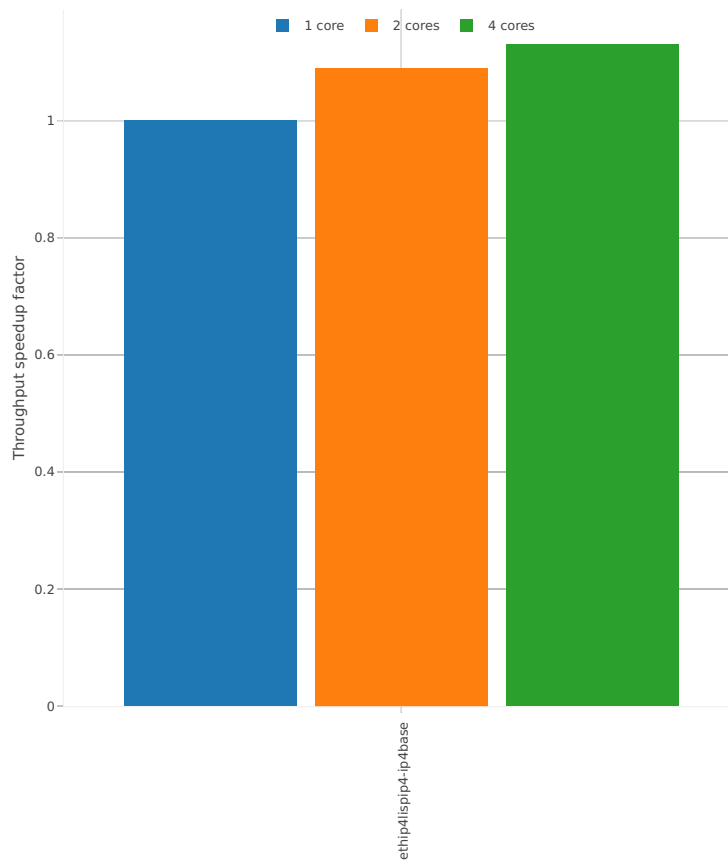
3n-skx-x710

64b-base_and_scale

Speedup: ip4tun-3n-skx-x710-64b-base_and_scale-ndr



Speedup: ip4tun-3n-skx-x710-64b-base_and_scale-pdr



2.4.6 IPv6 Tunnels

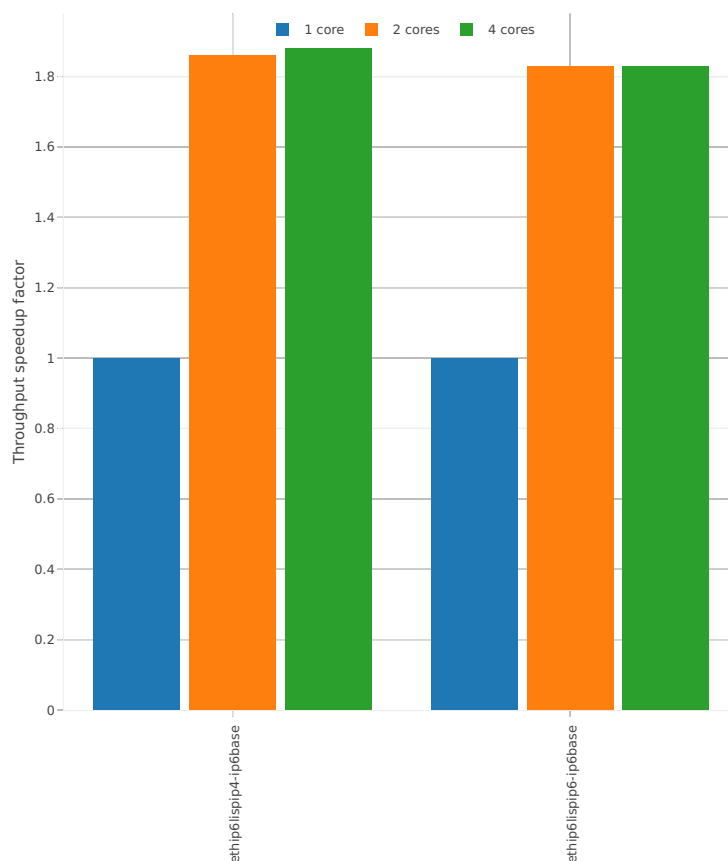
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Performance is reported for VPP 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](#)⁴⁹.

3n-hsw-x520

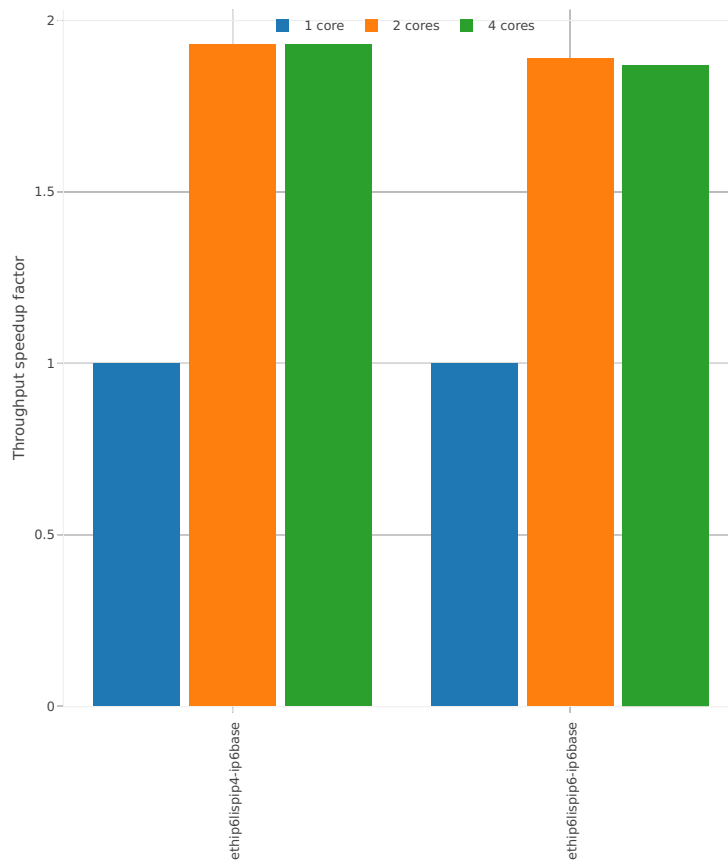
78b-base_and_scale

Speedup: ip6tun-3n-hsw-x520-78b-base_and_scale-ndr



⁴⁹ https://git.fd.io/csit/tree/tests/vpp/perf/ip6_tunnels?h=rls1807

Speedup: ip6tun-3n-hsw-x520-78b-base_and_scale-pdr



2.4.7 KVM VMs vhost-user

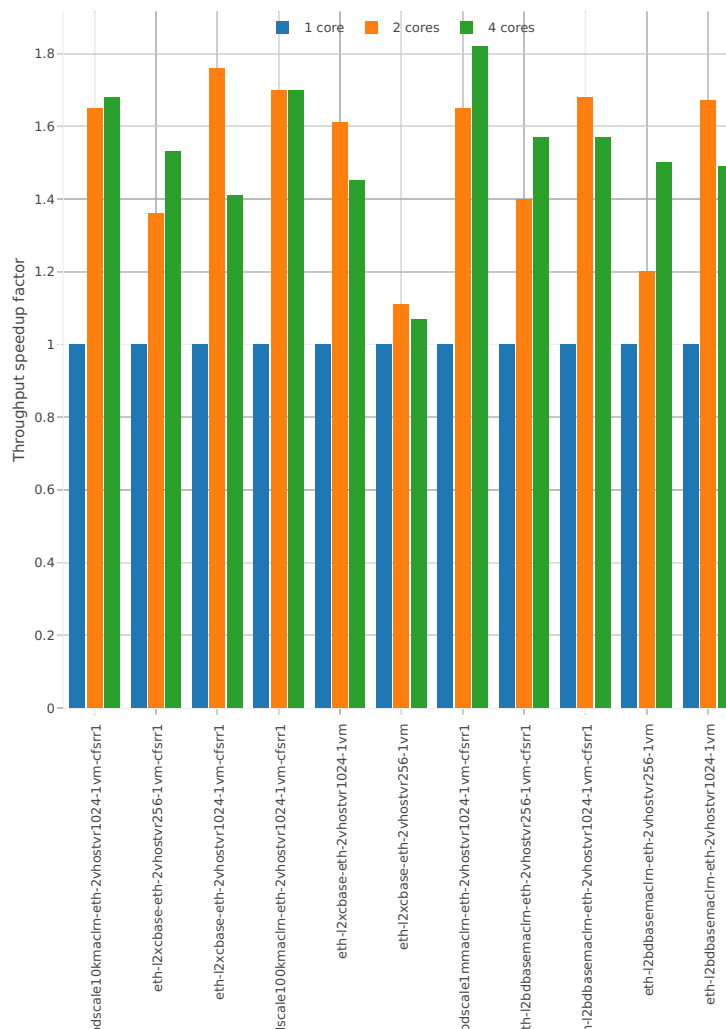
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 64B performance tests with VM vhost-user, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁵⁰.

3n-hsw-x520

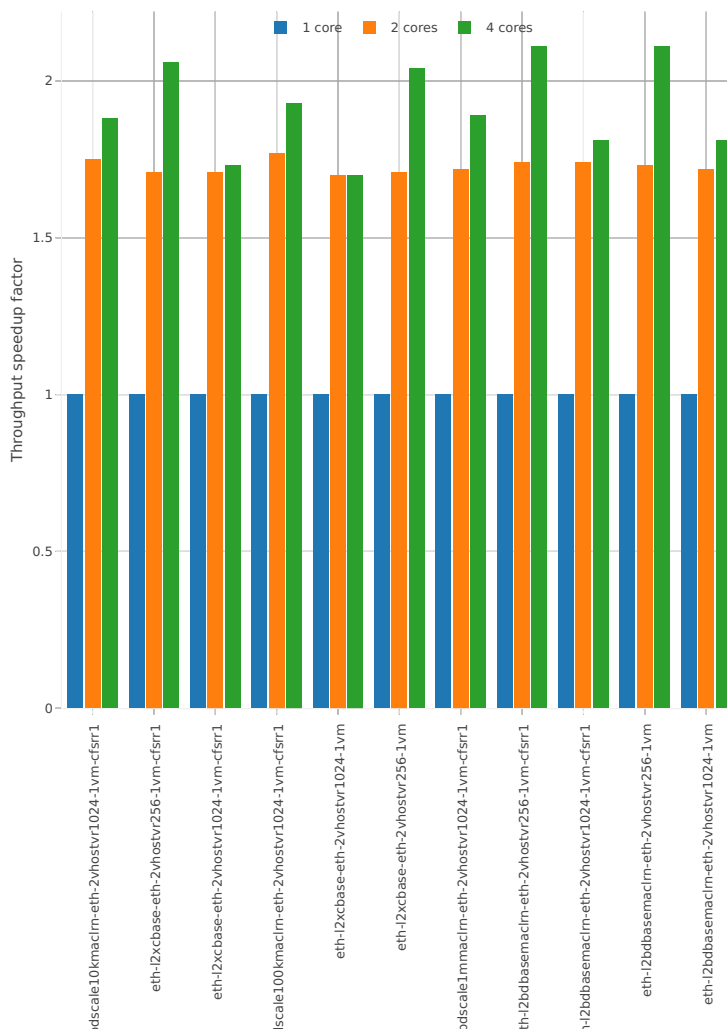
64b-base_and_scale-l2sw

Speedup: vhost-l2sw-3n-hsw-x520-64b-base_and_scale-ndr



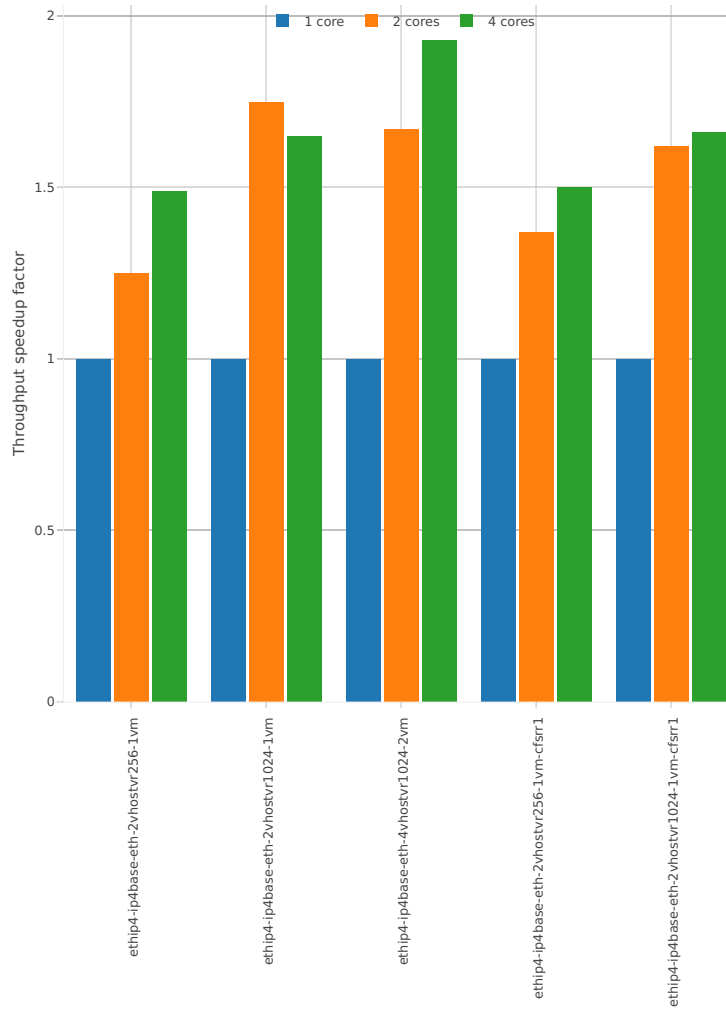
⁵⁰ https://git.fd.io/csit/tree/tests/vpp/perf/vm_vhost?h=rls1807

Speedup: vhost-l2sw-3n-hsw-x520-64b-base_and_scale-pdr

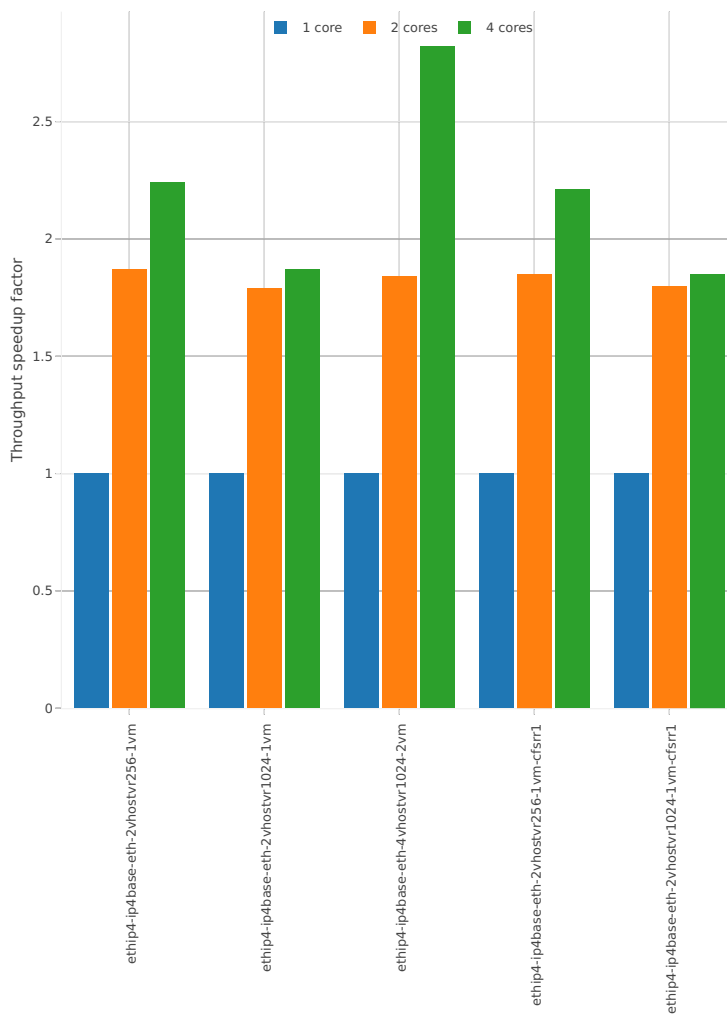


64b-base_and_scale-ip4

Speedup: vhost-ip4-3n-hsw-x520-64b-base_and_scale-ndr



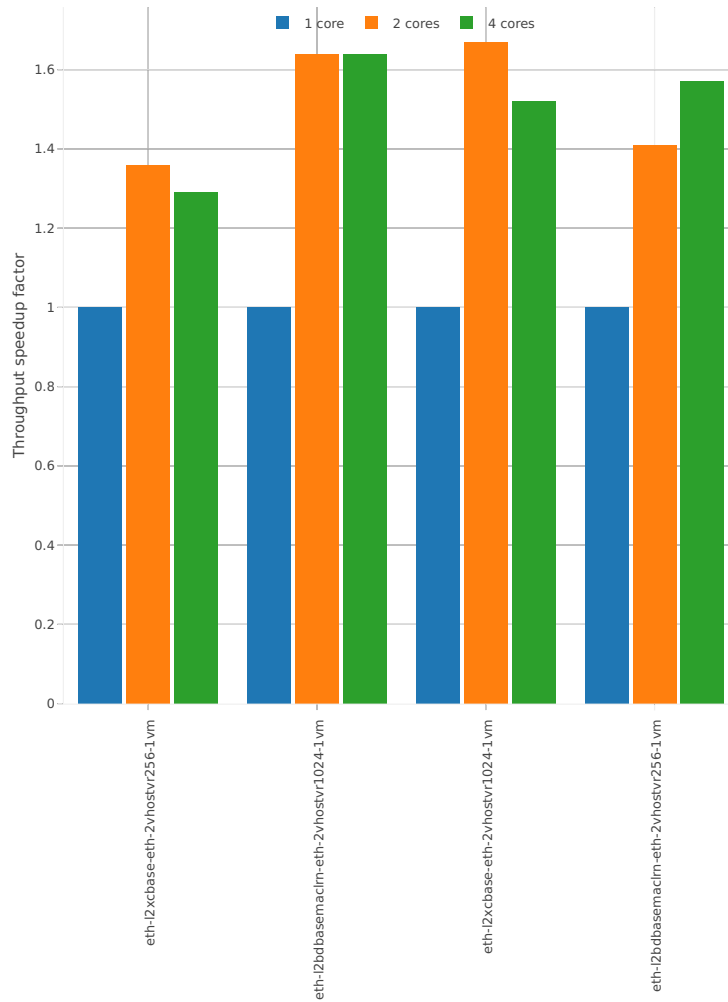
Speedup: vhost-ip4-3n-hsw-x520-64b-base_and_scale-pdr



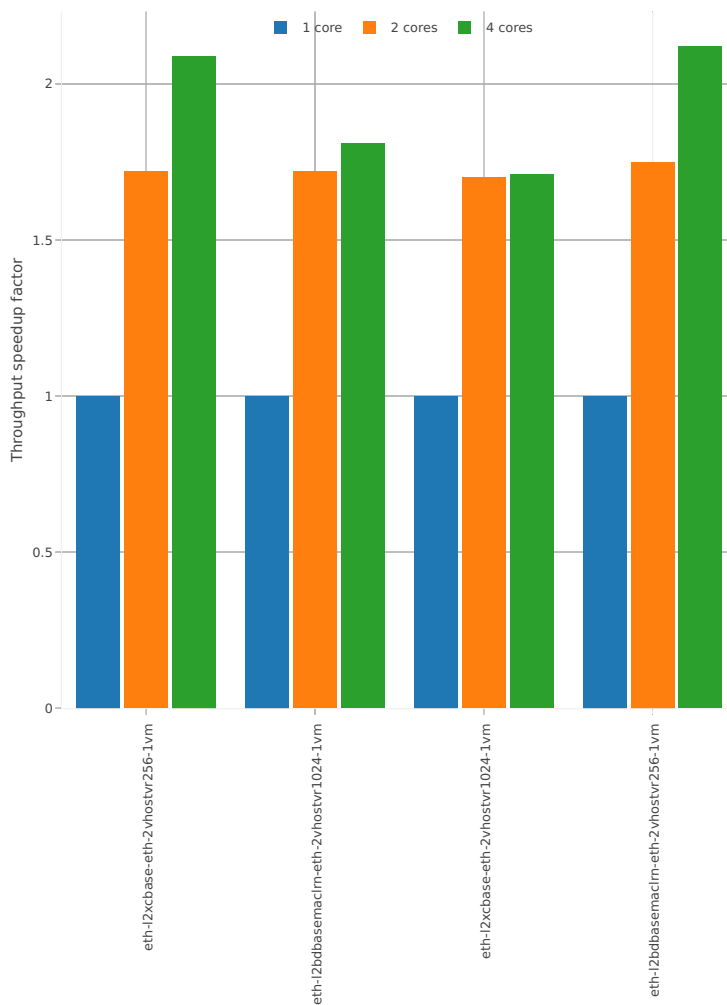
3n-hsw-x710

64b-base_and_scale-l2sw

Speedup: vhost-l2sw-3n-hsw-x710-64b-base_and_scale-ndr

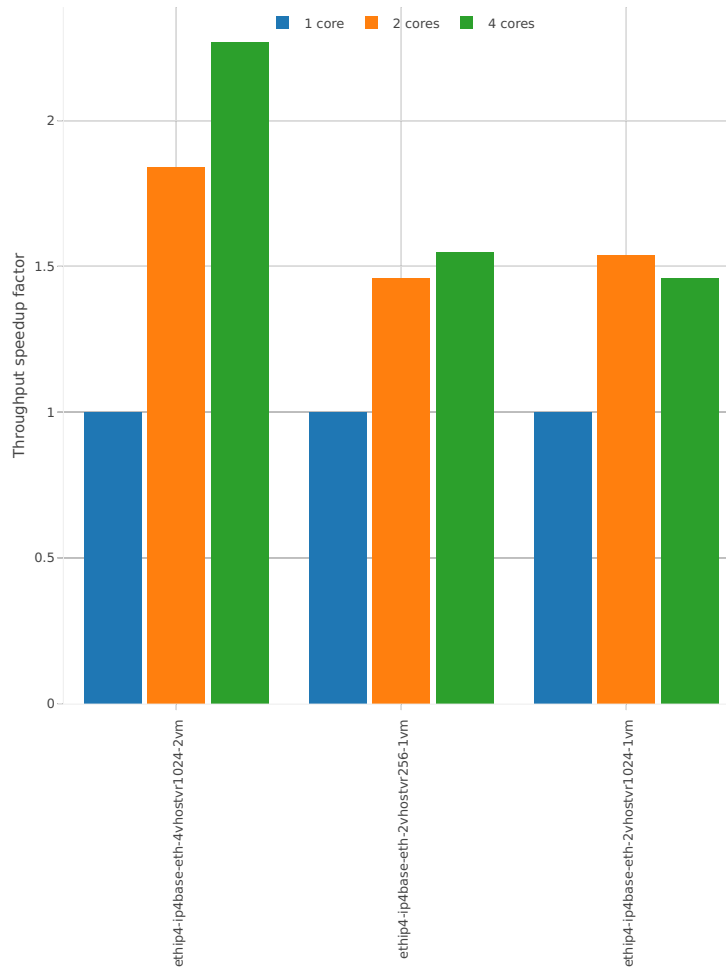


Speedup: vhost-l2sw-3n-hsw-x710-64b-base_and_scale-pdr

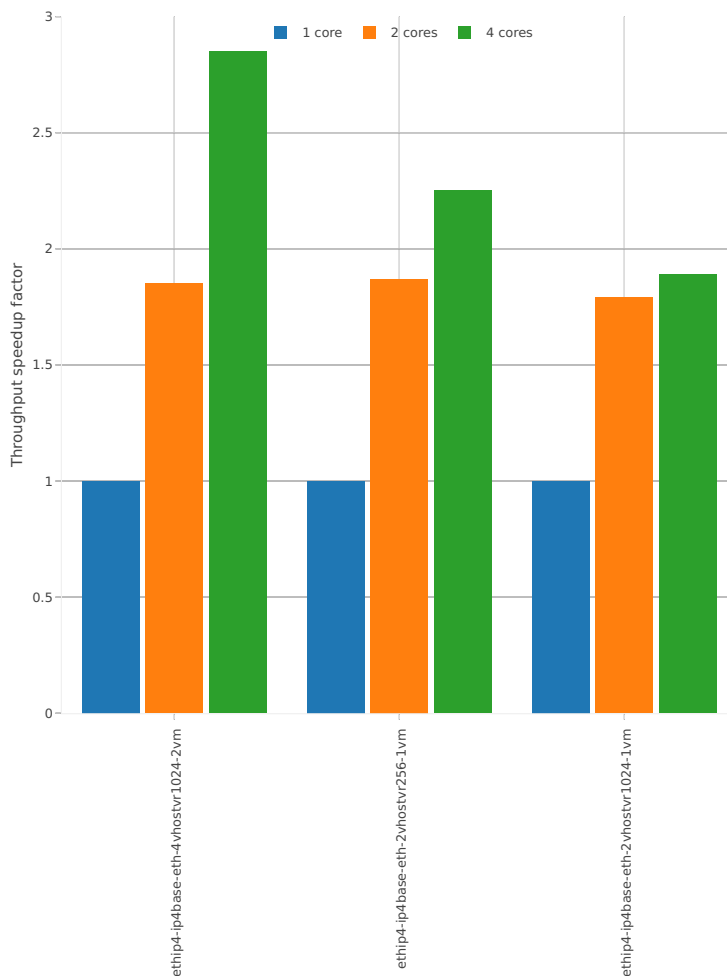


64b-base_and_scale-ip4

Speedup: vhost-ip4-3n-hsw-x710-64b-base_and_scale-ndr

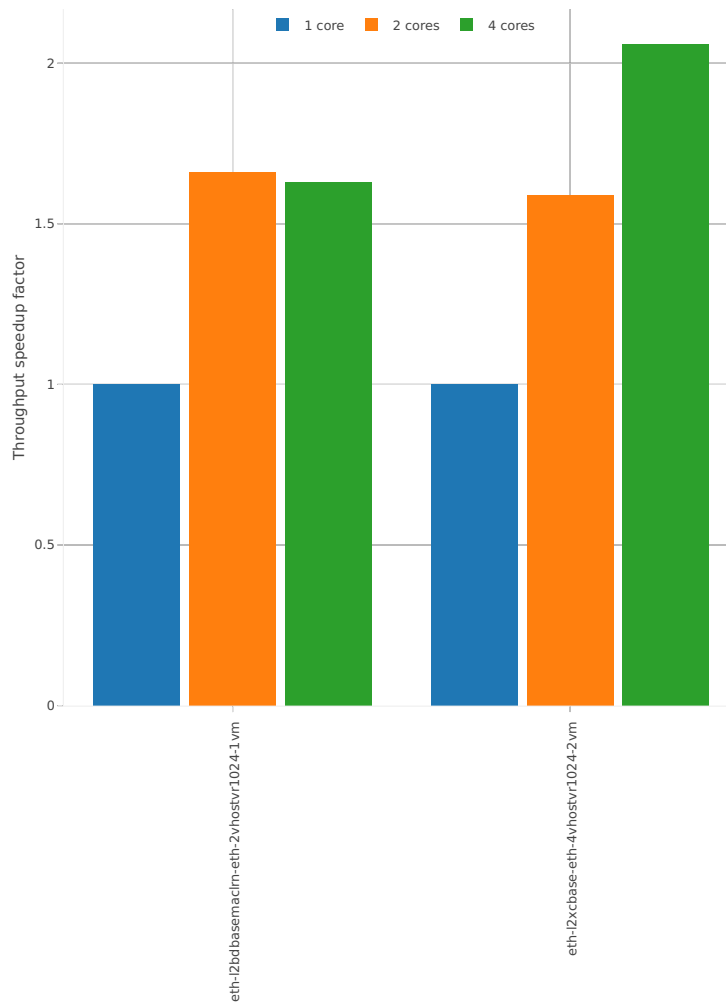


Speedup: vhost-ip4-3n-hsw-x710-64b-base_and_scale-pdr

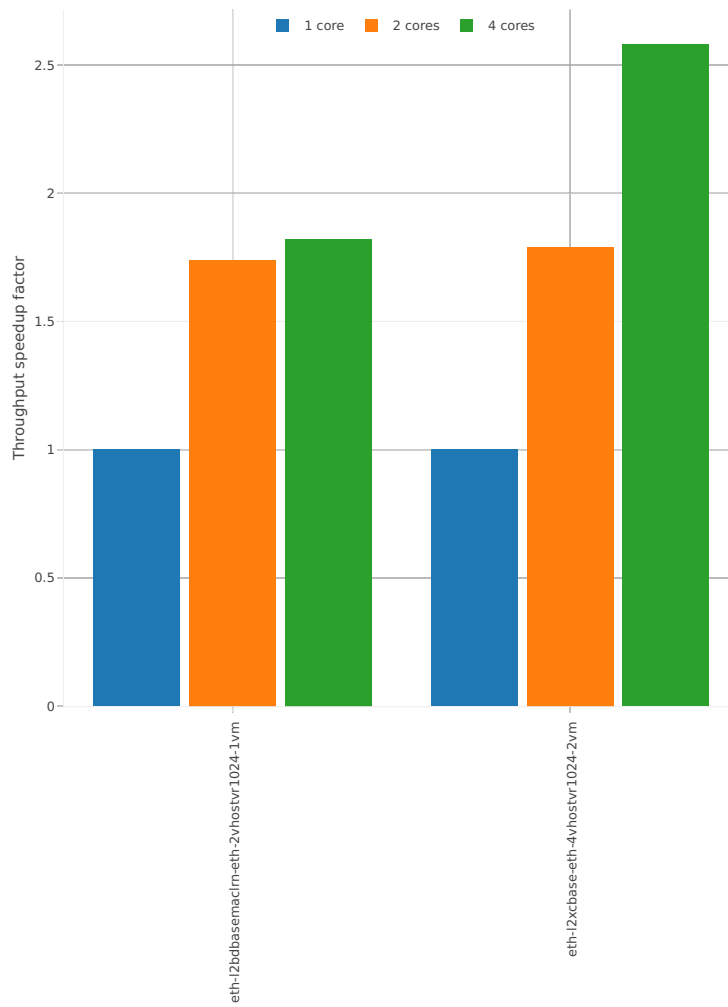


3n-hsw-xl710**64b-base_and_scale-l2sw**

Speedup: vhost-l2sw-3n-hsw-xl710-64b-base_and_scale-ndr

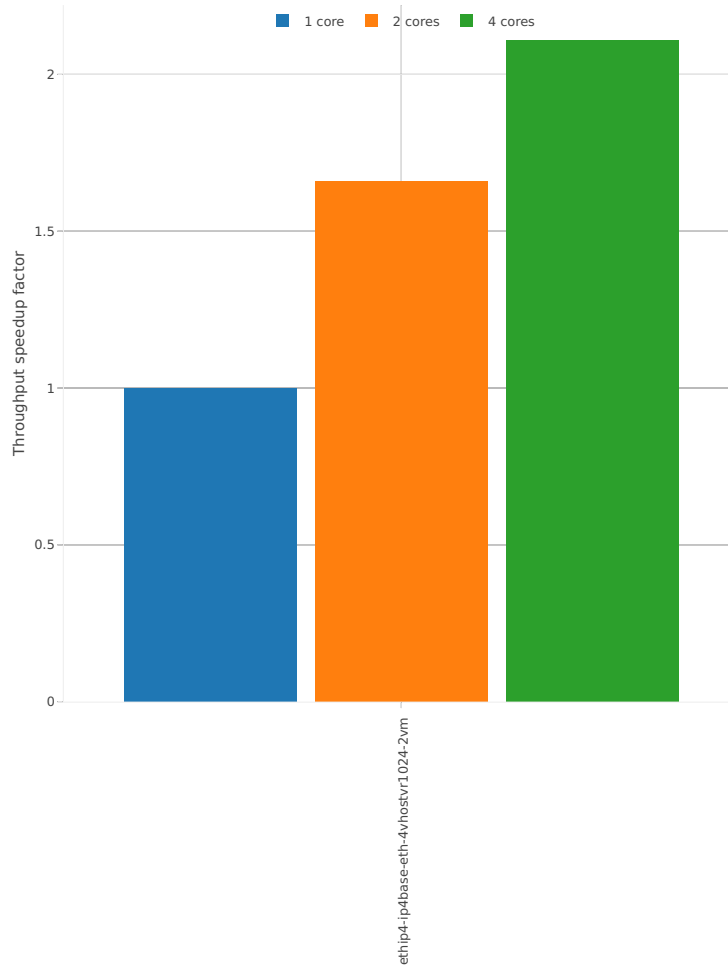


Speedup: vhost-l2sw-3n-hsw-xl710-64b-base_and_scale-pdr

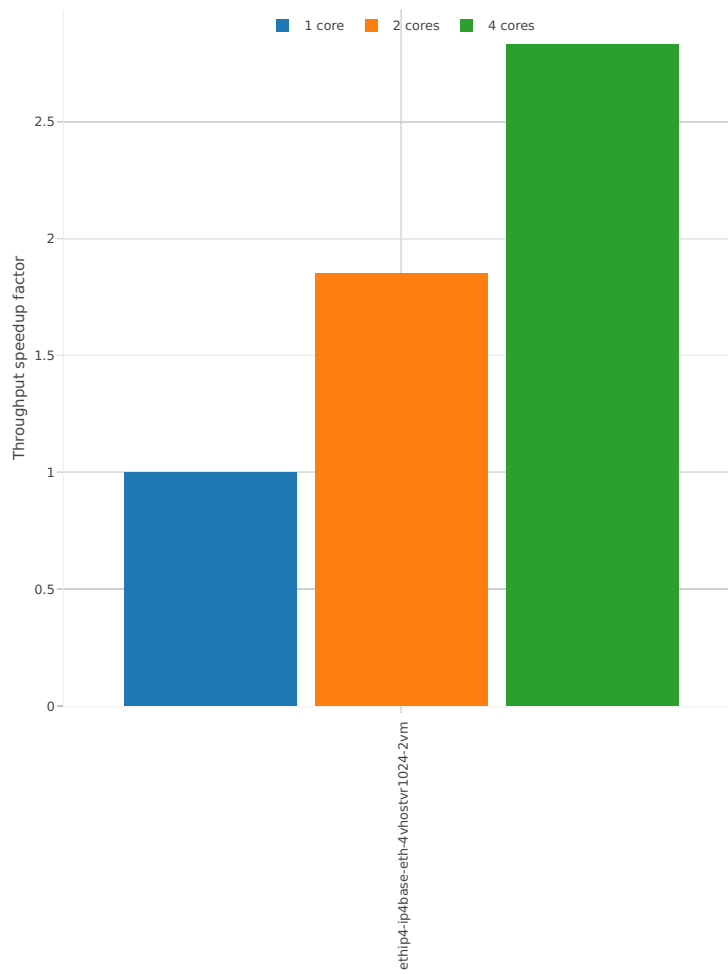


64b-base_and_scale-ip4

Speedup: vhost-ip4-3n-hsw-xl710-64b-base_and_scale-ndr



Speedup: vhost-ip4-3n-hsw-xl710-64b-base_and_scale-pdr



2.4.8 LXC/DRC Container Memif

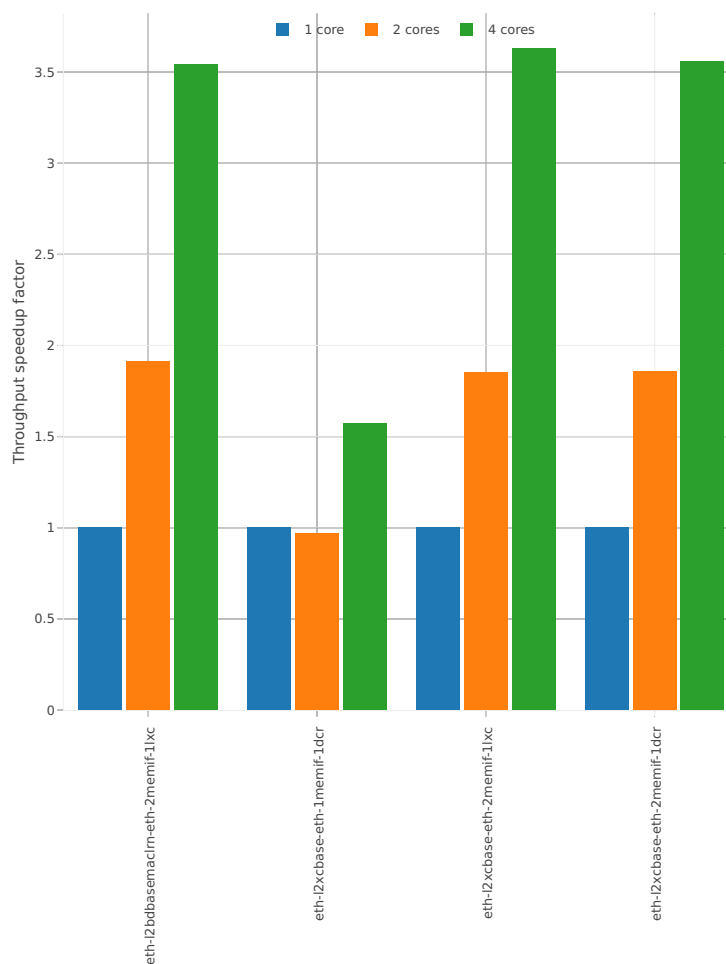
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Performance is reported for VPP 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](#)⁵¹.

3n-hsw-x520

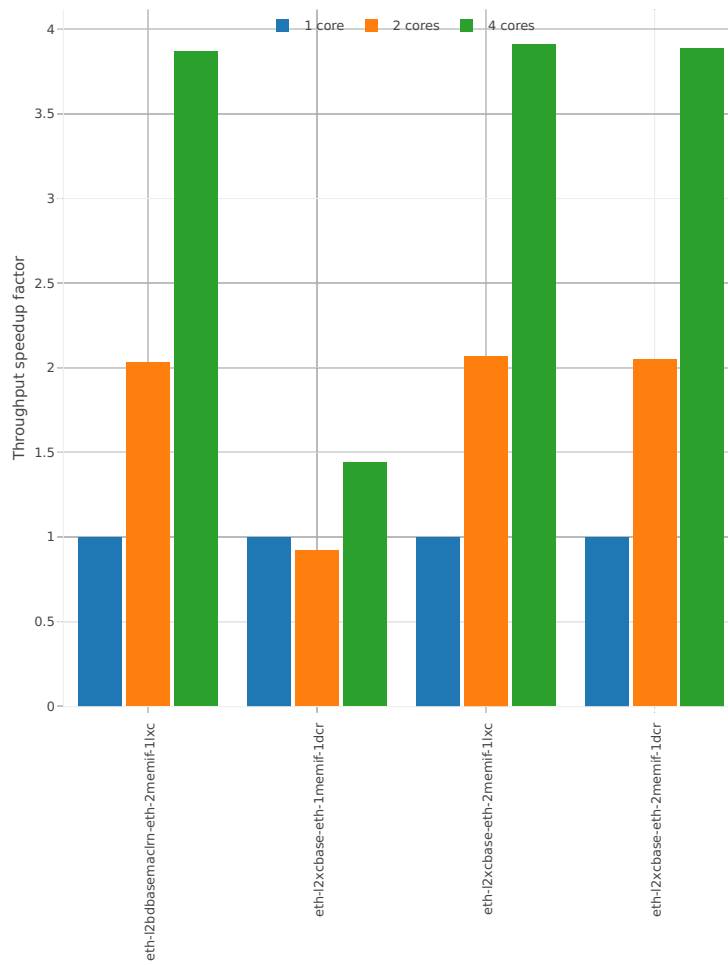
64b-base_and_scale

Speedup: memif-3n-hsw-x520-64b-base_and_scale-ndr



⁵¹ https://git.fd.io/csit/tree/tests/vpp/perf/container_memif?h=rls1807

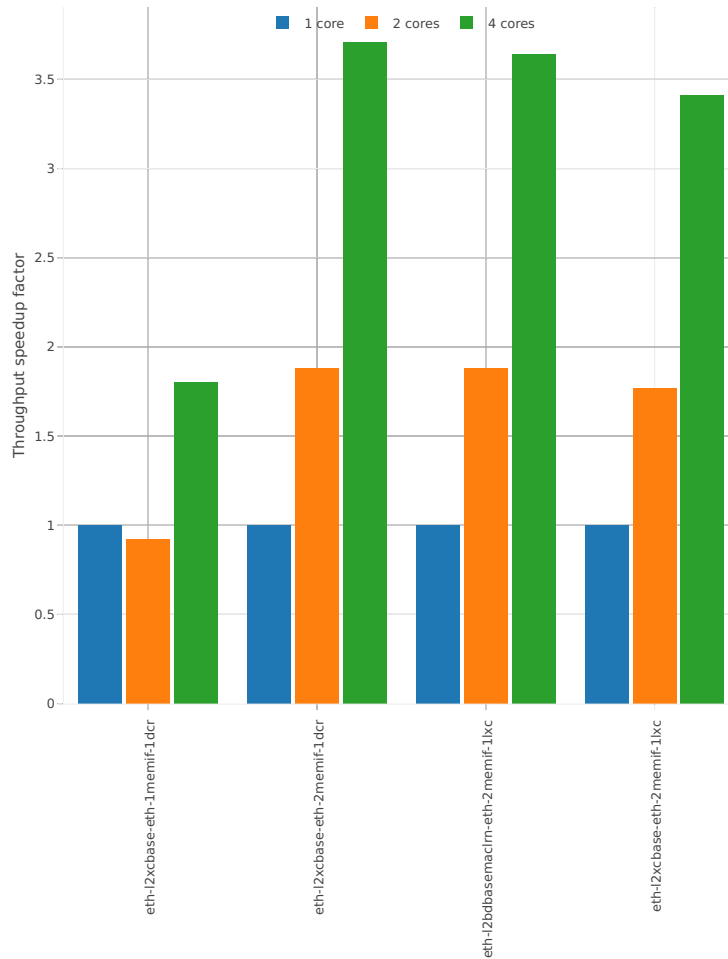
Speedup: memif-3n-hsw-x520-64b-base_and_scale-pdr



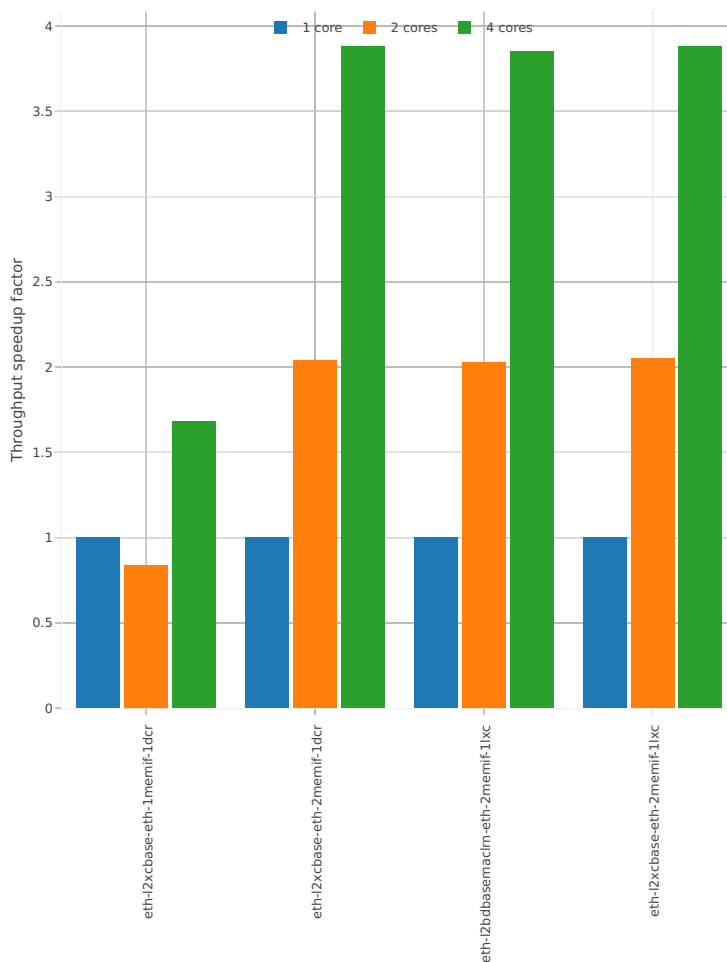
3n-hsw-x710

64b-base_and_scale

Speedup: memif-3n-hsw-x710-64b-base_and_scale-ndr



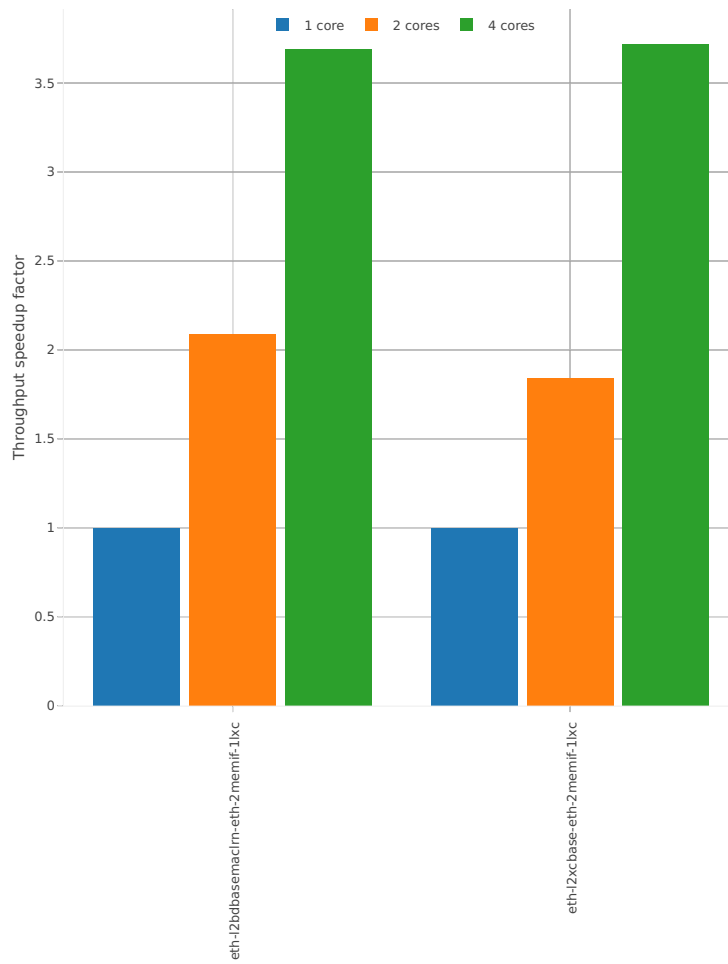
Speedup: memif-3n-hsw-x710-64b-base_and_scale-pdr



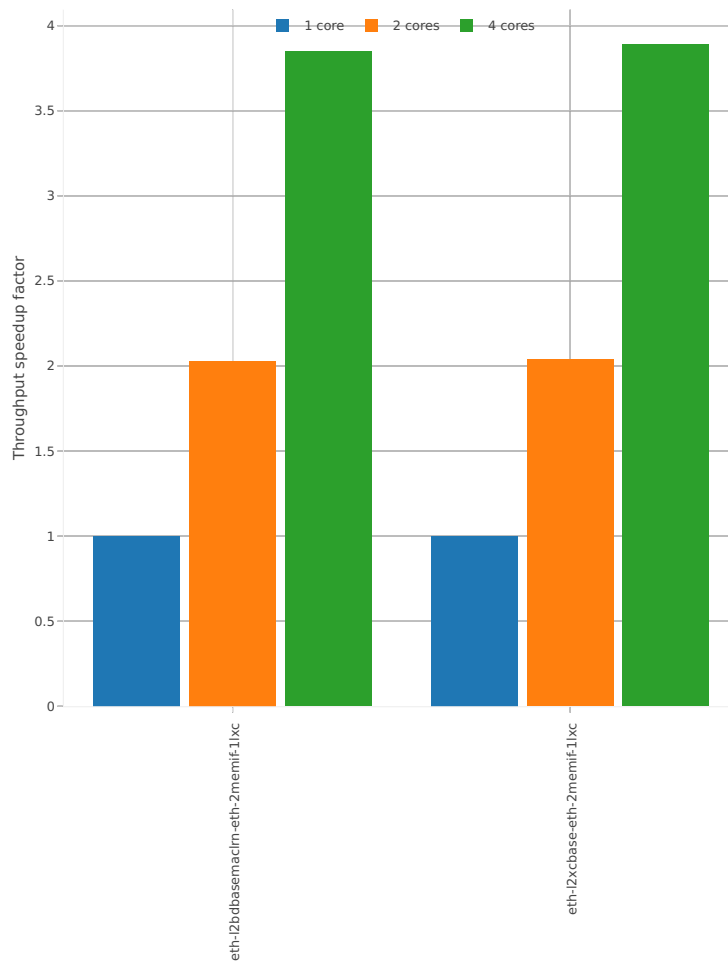
3n-hsw-xl710

64b-base_and_scale

Speedup: memif-3n-hsw-xl710-64b-base_and_scale-ndr



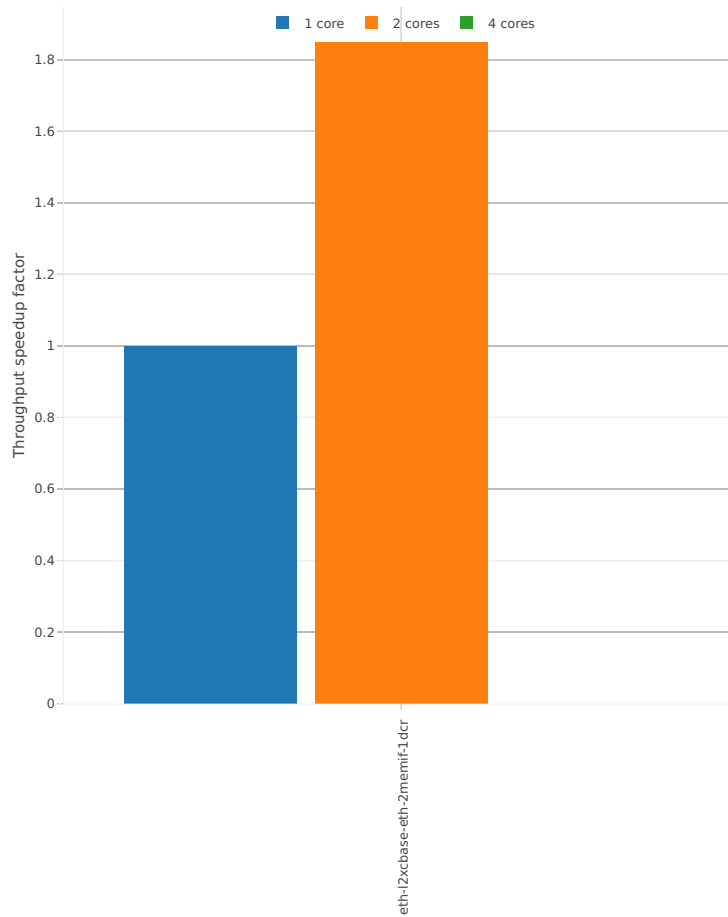
Speedup: memif-3n-hsw-xl710-64b-base_and_scale-pdr



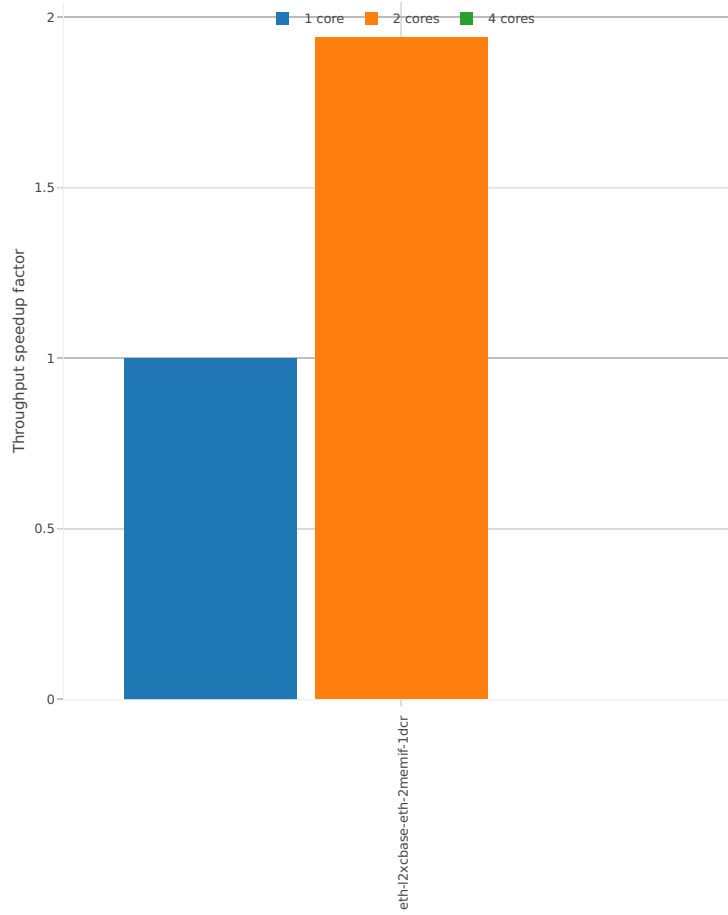
3n-skx-x710

64b-base_and_scale

Speedup: memif-3n-skx-x710-64b-base_and_scale-ndr



Speedup: memif-3n-skx-x710-64b-base_and_scale-pdr



2.4.9 K8s Container Memif

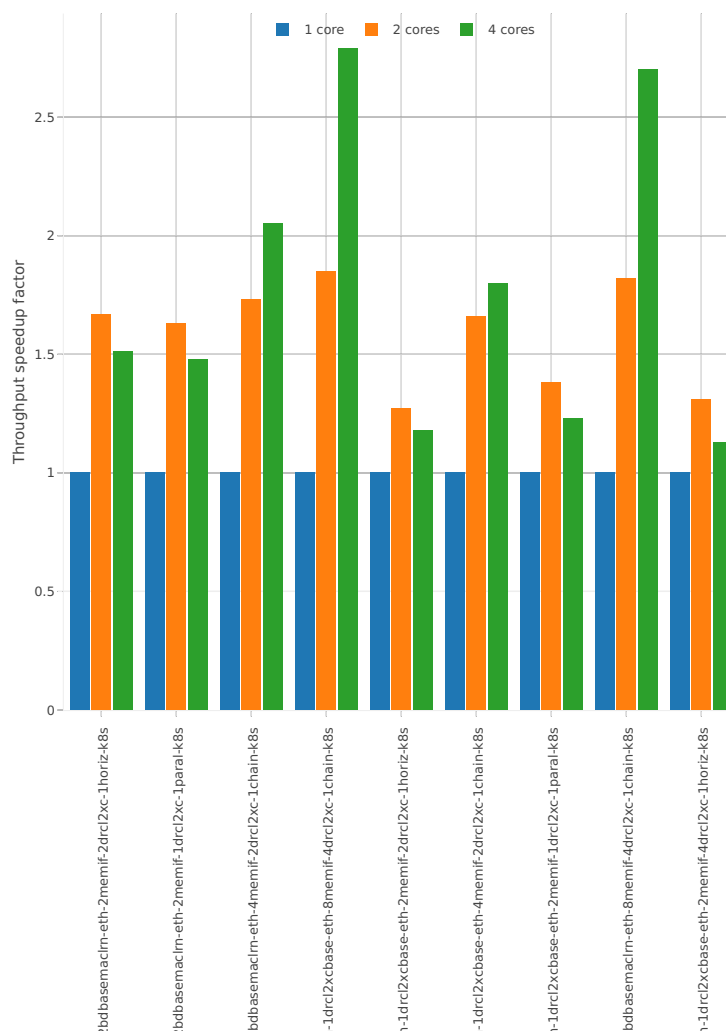
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Performance is reported for VPP 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](#)⁵².

3n-hsw-x520

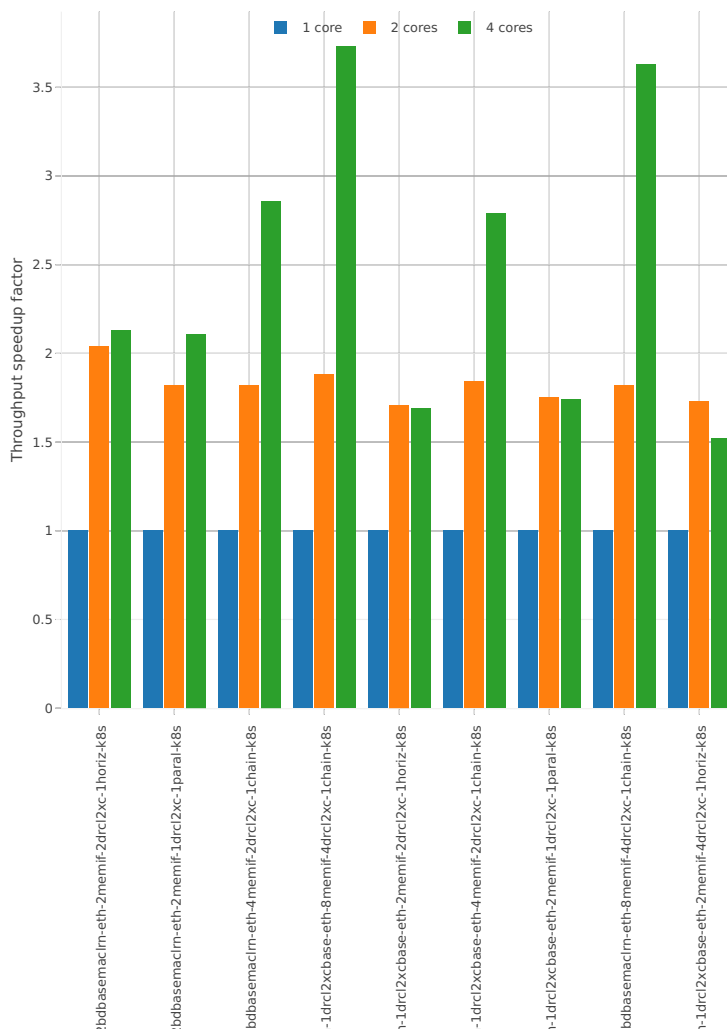
64b-base_and_scale

Speedup: k8s-memif-3n-hsw-x520-64b-base_and_scale-ndr



⁵² https://git.fd.io/csit/tree/tests/kubernetes/perf/container_memif?h=rls1807

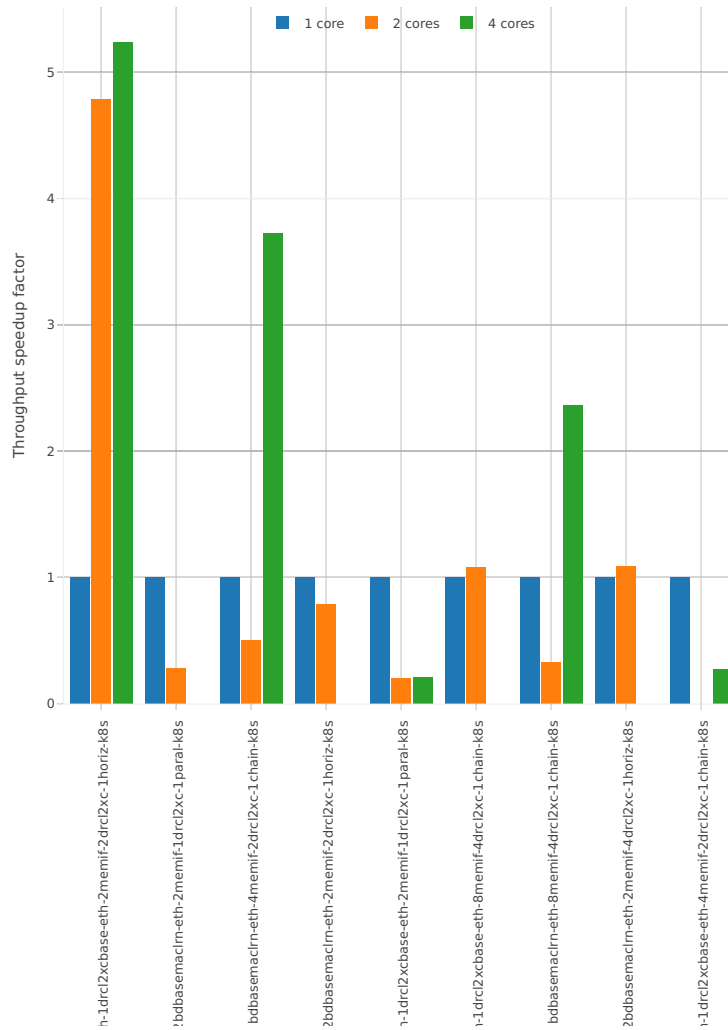
Speedup: k8s-memif-3n-hsw-x520-64b-base_and_scale-pdr



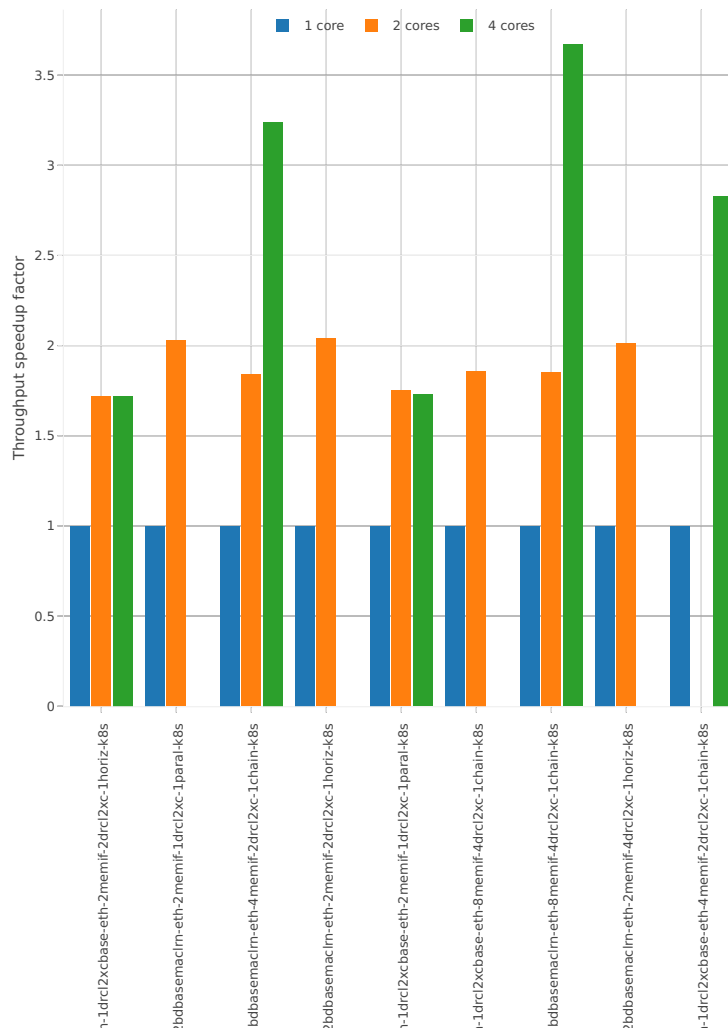
3n-hsw-x710

64b-base_and_scale

Speedup: k8s-memif-3n-hsw-x710-64b-base_and_scale-ndr



Speedup: k8s-memif-3n-hsw-x710-64b-base_and_scale-pdr



2.4.10 IPsec IPv4 Routing

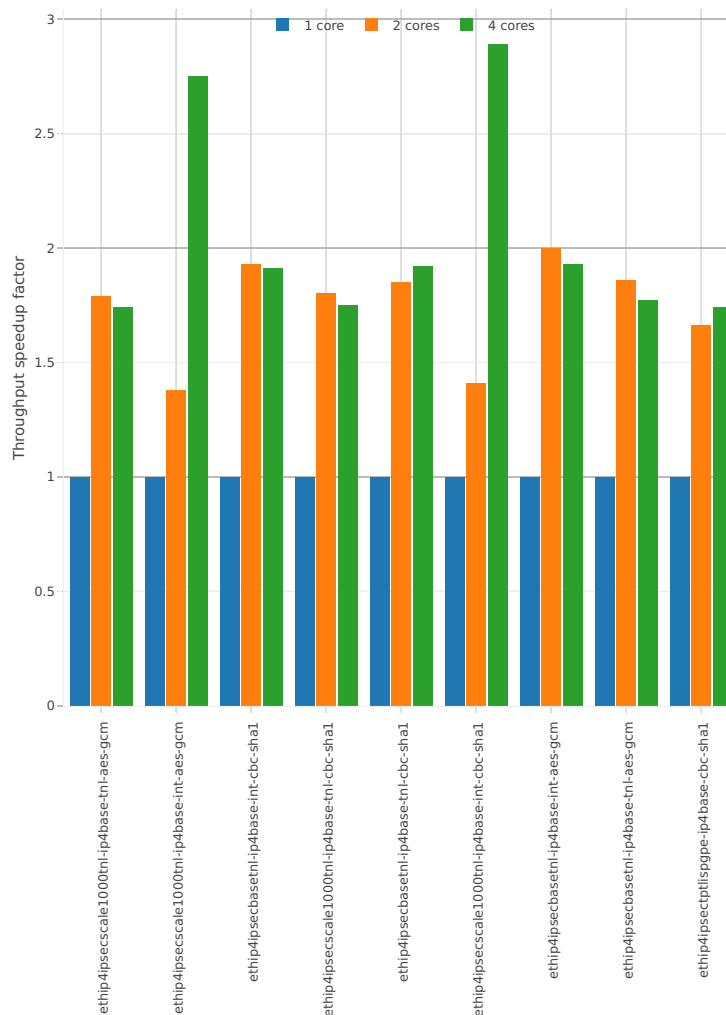
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. VPP IPsec encryption is accelerated 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](#)⁵³.

3n-hsw-xl710

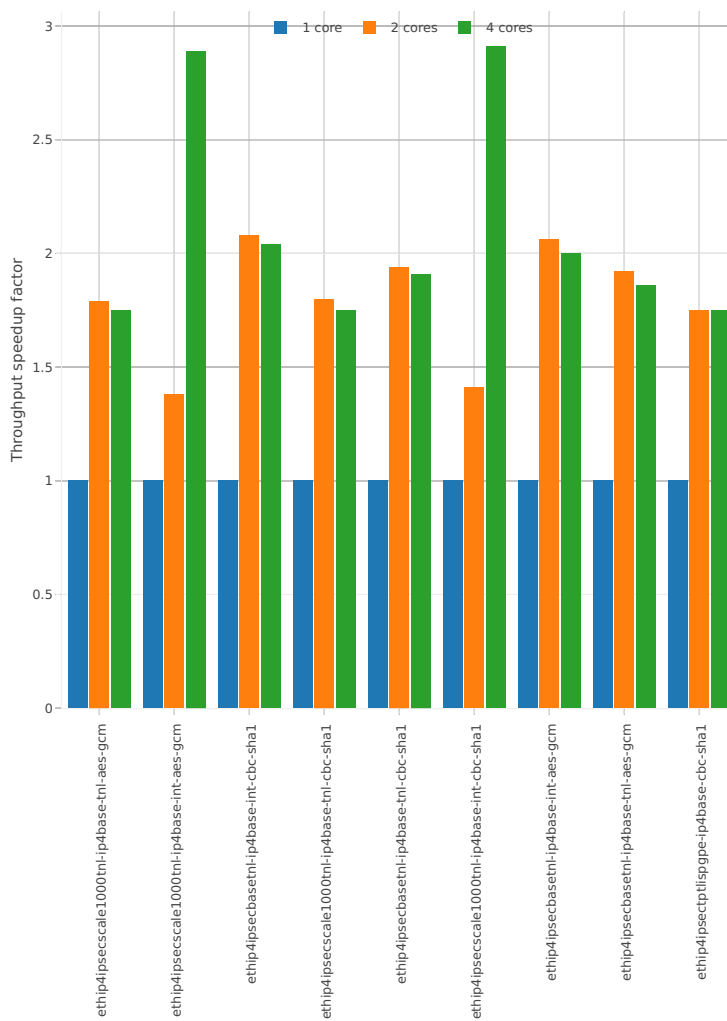
64b-base_and_scale

Speedup: ipsec-3n-hsw-xl710-64b-base_and_scale-ndr



⁵³ <https://git.fd.io/csit/tree/tests/vpp/perf/crypto?h=rls1807>

Speedup: ipsec-3n-hsw-xl710-64b-base_and_scale-pdr



2.4.11 VTS

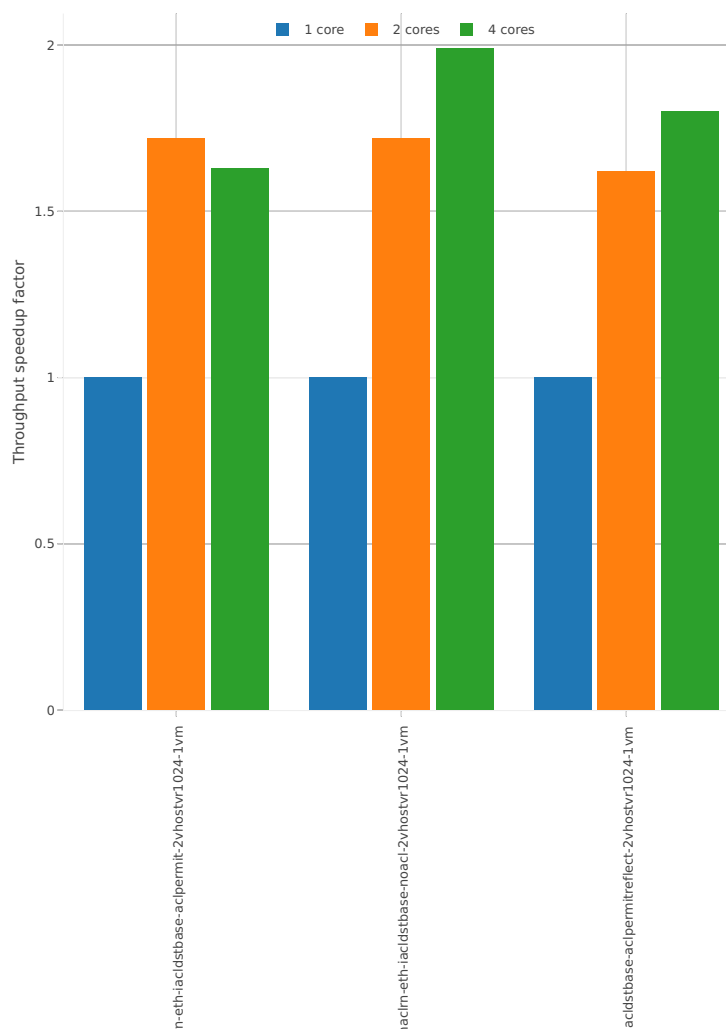
Following sections include Throughput Speedup Analysis for VPP multi-core multi-thread configurations with no Hyper-Threading, specifically for tested 2t2c (2threads, 2cores) and 4t4c scenarios. 1t1c throughput results are used as a reference for reported speedup ratio. Input data used for the graphs comes from Phy-to-Phy 64B performance tests with VM vhost-user, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss).

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁵⁴.

3n-hsw-x520

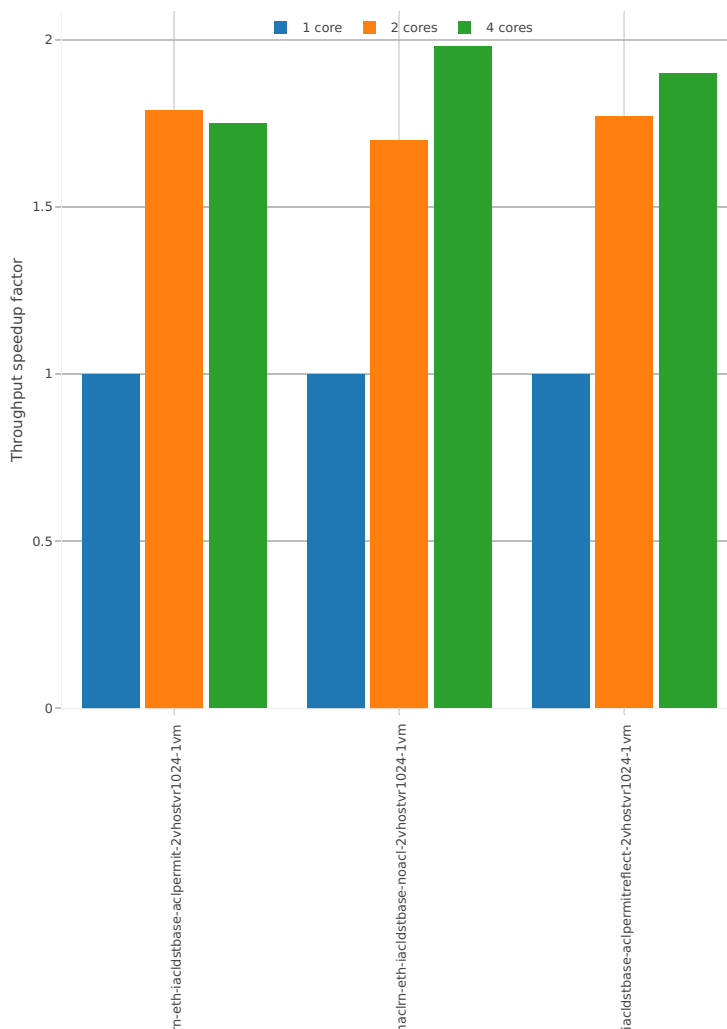
114b-base_and_scale

Speedup: vts-3n-hsw-x520-114b-base_and_scale-ndr



⁵⁴ <https://git.fd.io/csit/tree/tests/vpp/perf/vts?h=rls1807>

Speedup: vts-3n-hsw-x520-114b-base_and_scale-pdr



2.5 Packet Latency

Latency results are generated from a single execution of NDR discovery test across physical testbeds hosted LF FD.io labs: 3n-hsw, 2n-skx, 2n- skx. Box plots are used to show the Minimum, Median and Maximum packet latency per test.

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 DUT configuration.
2. **X-axis Labels:** indices of individual test suites as listed in Graph Legend and direction of latency flow:
 - West-to-East: TGint1-to-SUT1-to-SUT2-to-TGint2.
 - East-to-West: TGint2-to-SUT2-to-SUT1-to-TGint1.
3. **Y-axis Labels:** measured packet latency values in [uSec].
4. **Graph Legend:** lists X-axis indices with associated CSIT test suites executed to generate graphed test results.

Note: Test results have been generated by [FD.io test executor vpp performance job 3n-hsw⁵⁵](#), [FD.io test executor vpp performance job 3n-skx⁵⁶](#) and [FD.io test executor vpp performance job 2n-skx⁵⁷](#) with RF result files csit-vpp-perf-1807-*.zip [archived here](#).

⁵⁵ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

⁵⁶ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-skx>

⁵⁷ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-2n-skx>

2.5.1 L2 Ethernet Switching

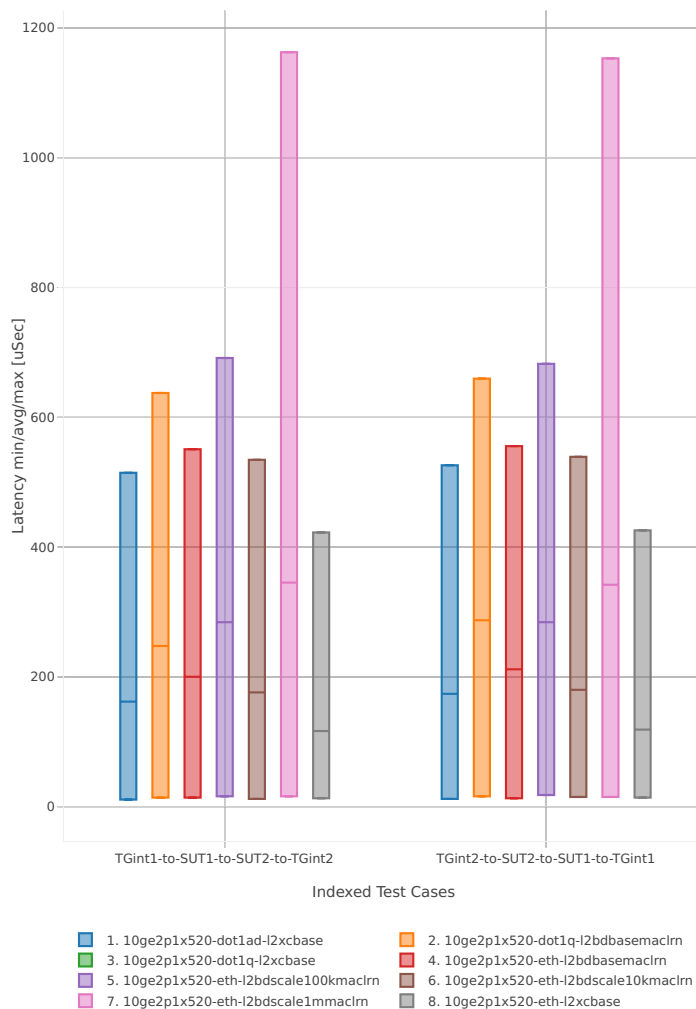
This section includes summary graphs of VPP Phy-to-Phy packet latency with L2 Ethernet switching measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁵⁸.

3n-hsw-x520

64b-1t1c-base_and_scale

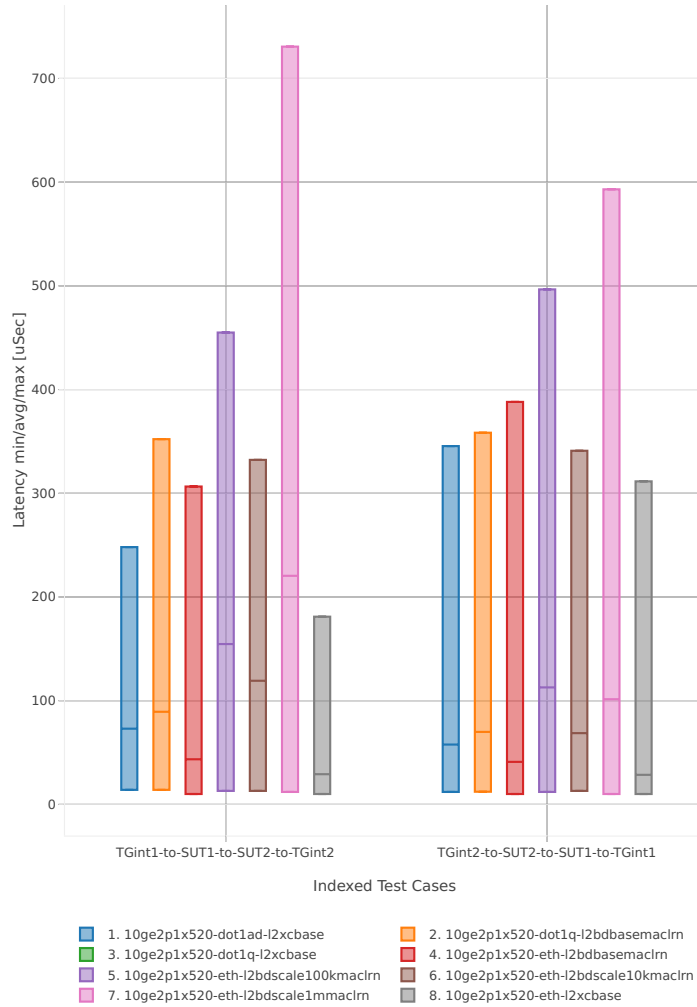
Latency: l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁵⁸ <https://git.fd.io/csit/tree/tests/vpp/perf/l2?h=rls1807>

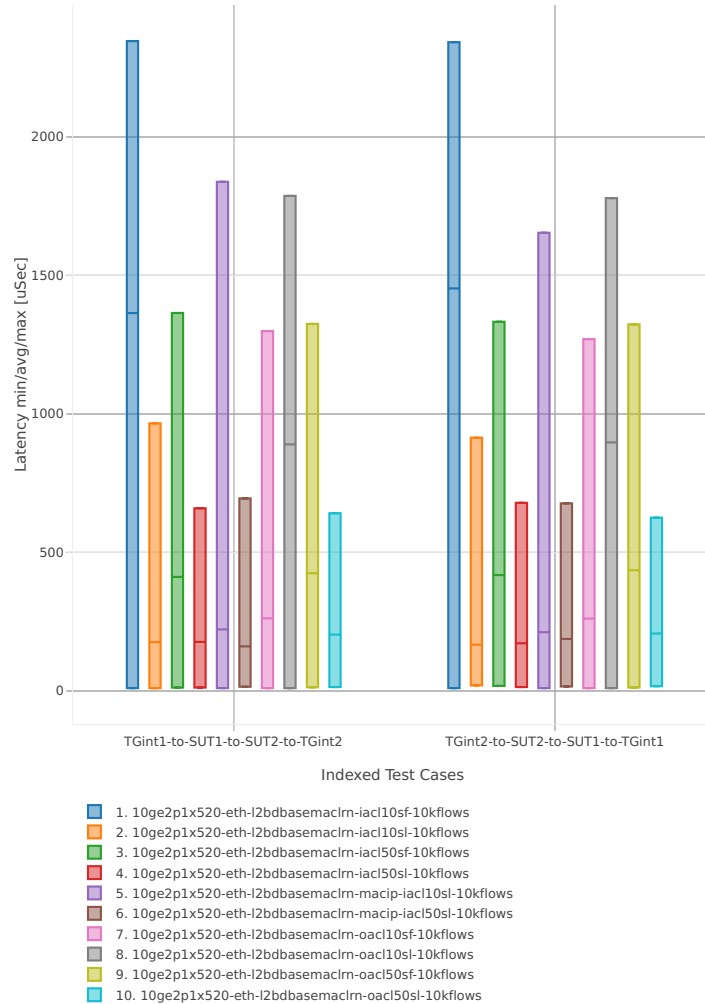
64b-2t2c-base_and_scale

Latency: l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



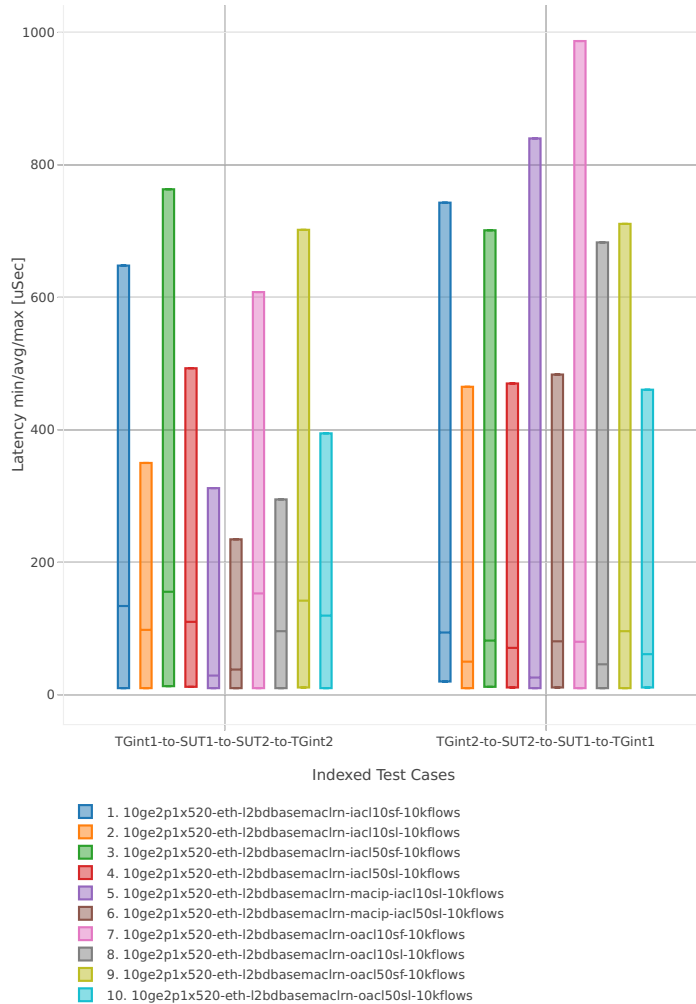
64b-1t1c-features

Latency: l2sw-3n-hsw-x520-64b-1t1c-features-ndr



64b-2t2c-features

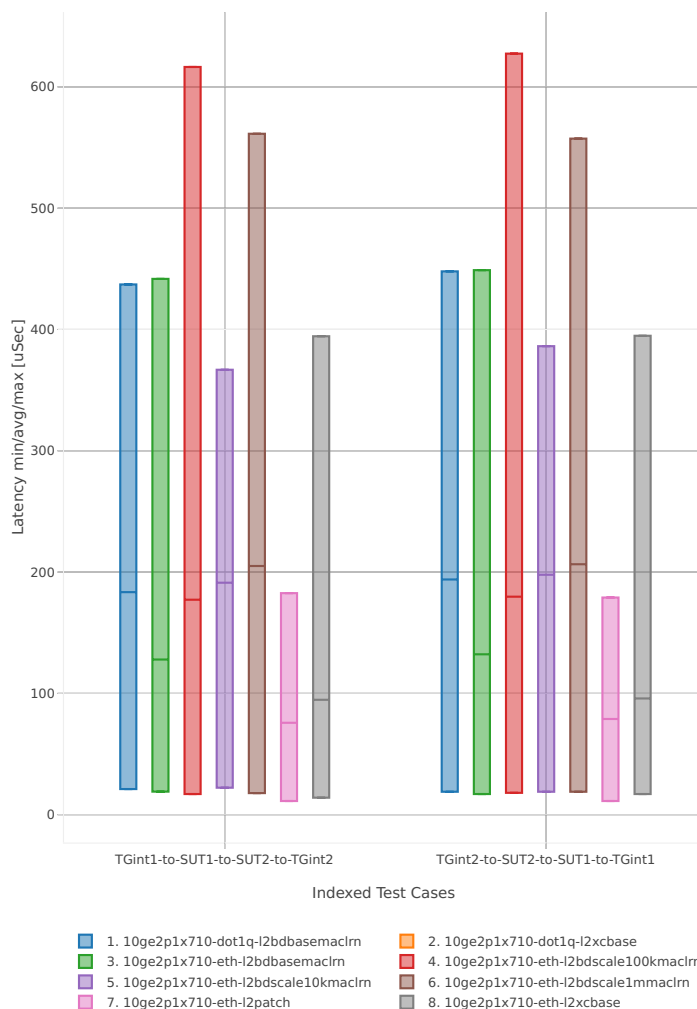
Latency: l2sw-3n-hsw-x520-64b-2t2c-features-ndr



3n-hsw-x710

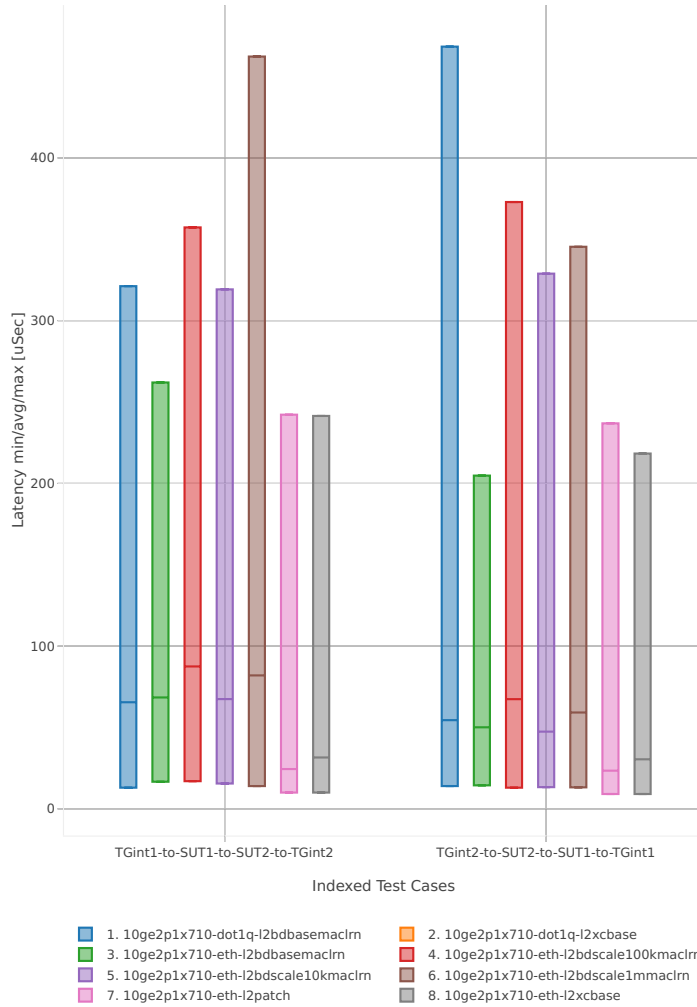
64b-1t1c-base_and_scale

Latency: l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



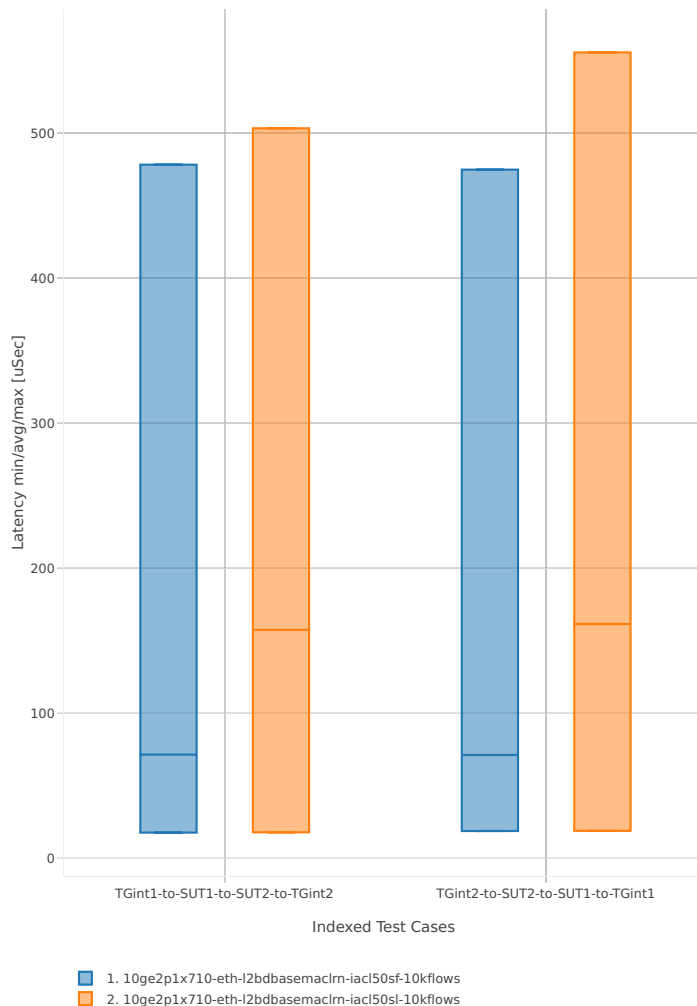
64b-2t2c-base_and_scale

Latency: l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



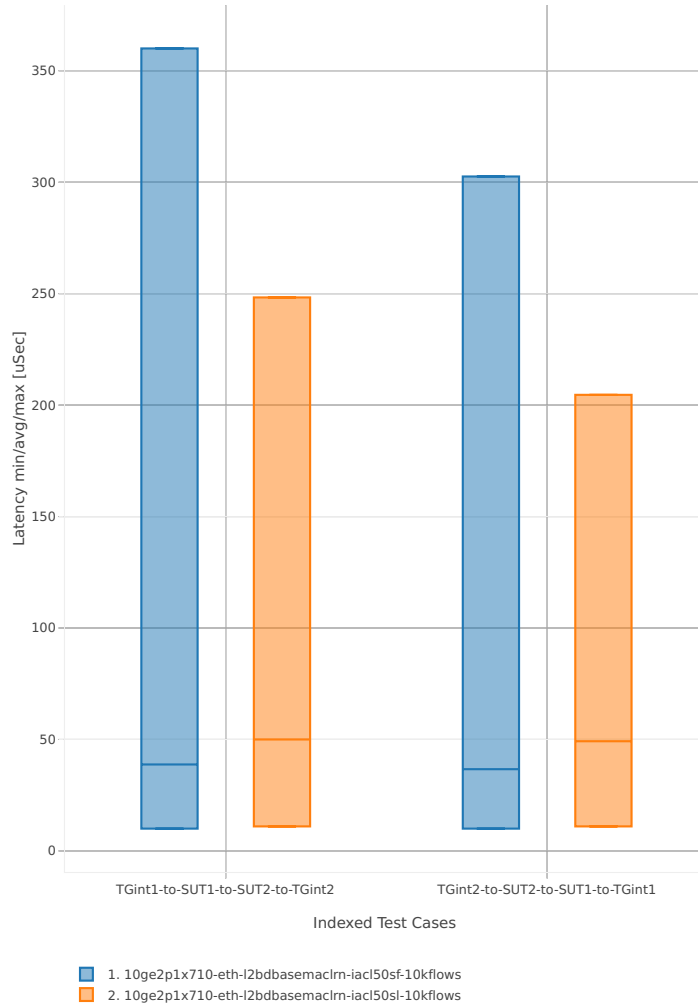
64b-1t1c-features

Latency: l2sw-3n-hsw-x710-64b-1t1c-features-ndr



64b-2t2c-features

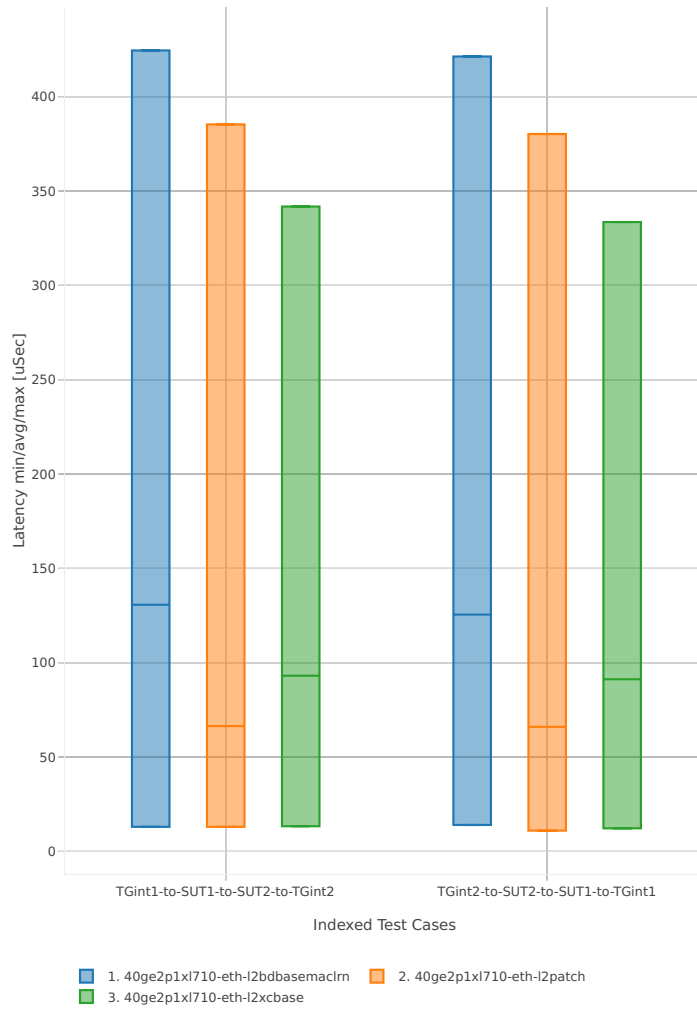
Latency: l2sw-3n-hsw-x710-64b-2t2c-features-ndr



3n-hsw-xl710

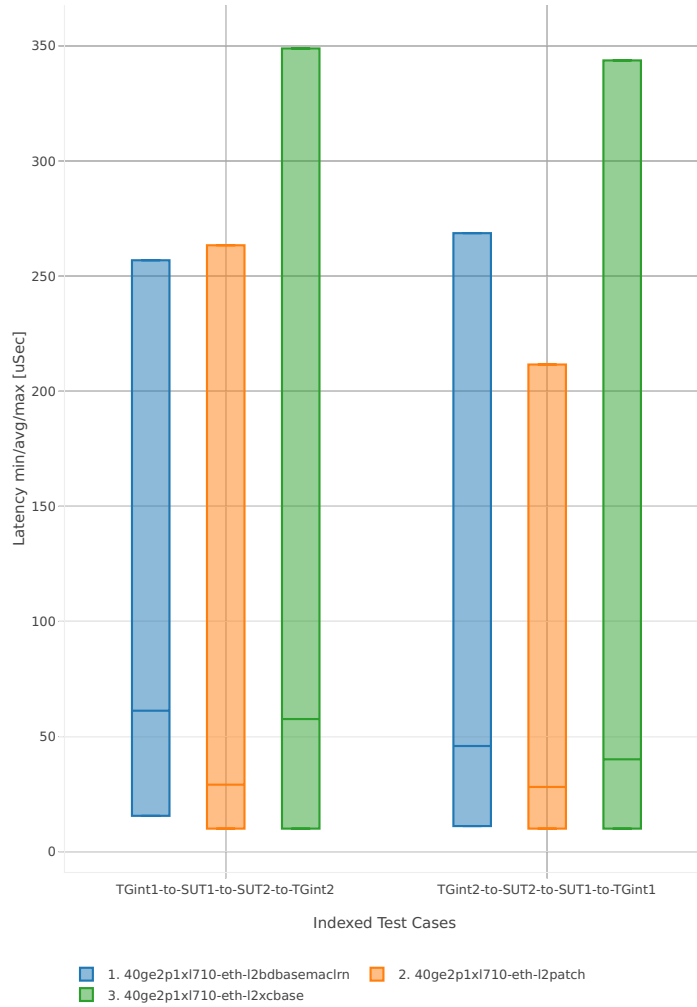
64b-1t1c-base_and_scale

Latency: l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

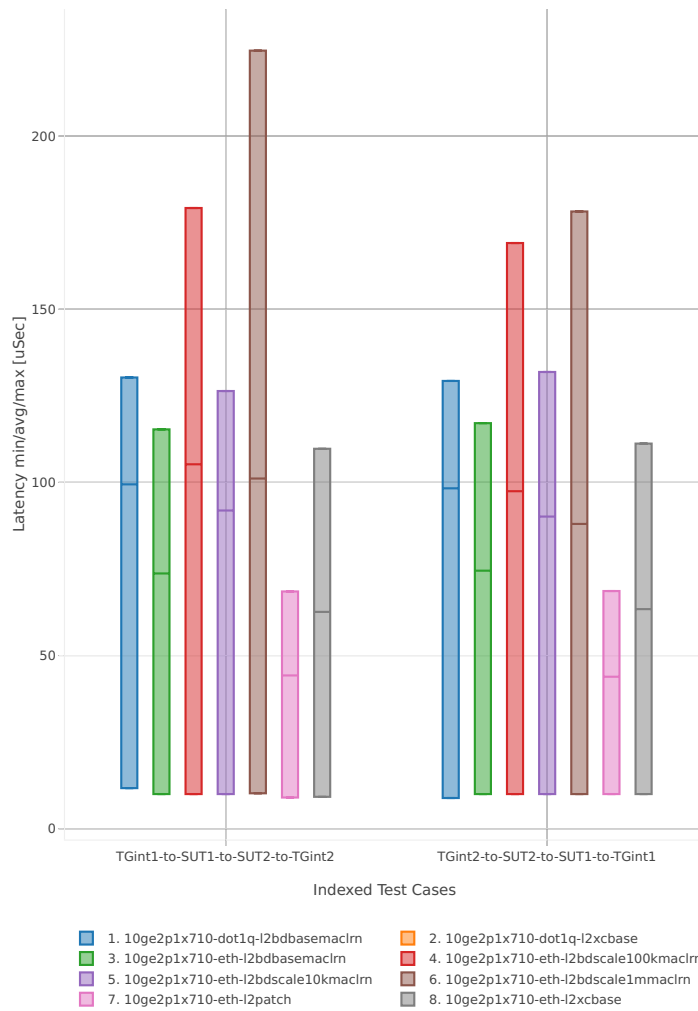
Latency: l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



3n-skx-x710

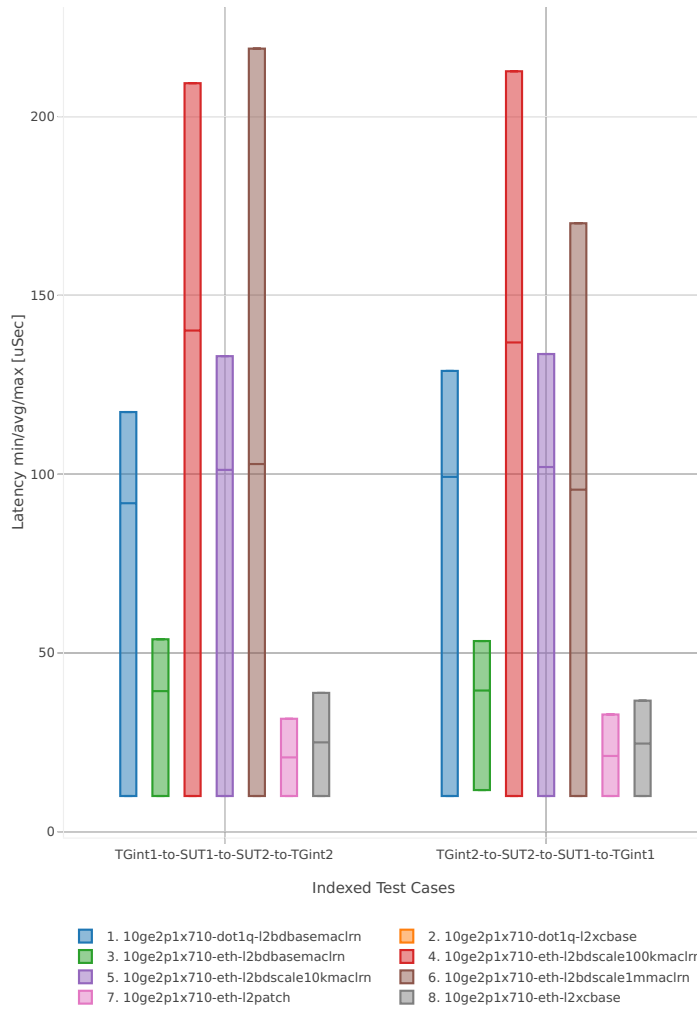
64b-2t1c-base_and_scale

Latency: l2sw-3n-skx-x710-64b-2t1c-base_and_scale-ndr



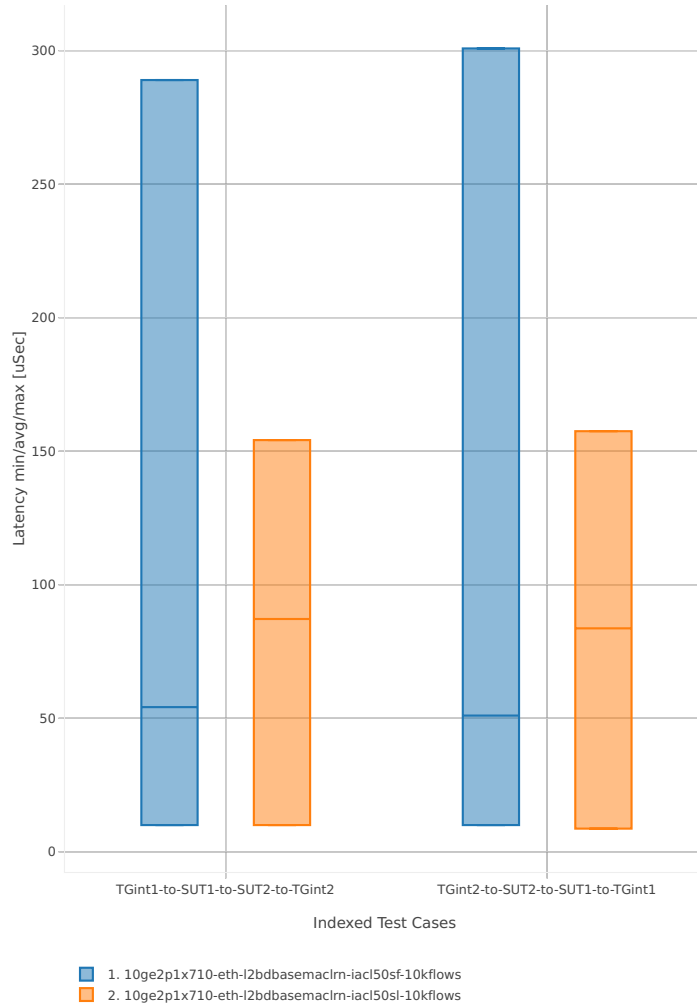
64b-4t2c-base_and_scale

Latency: l2sw-3n-skx-x710-64b-4t2c-base_and_scale-ndr



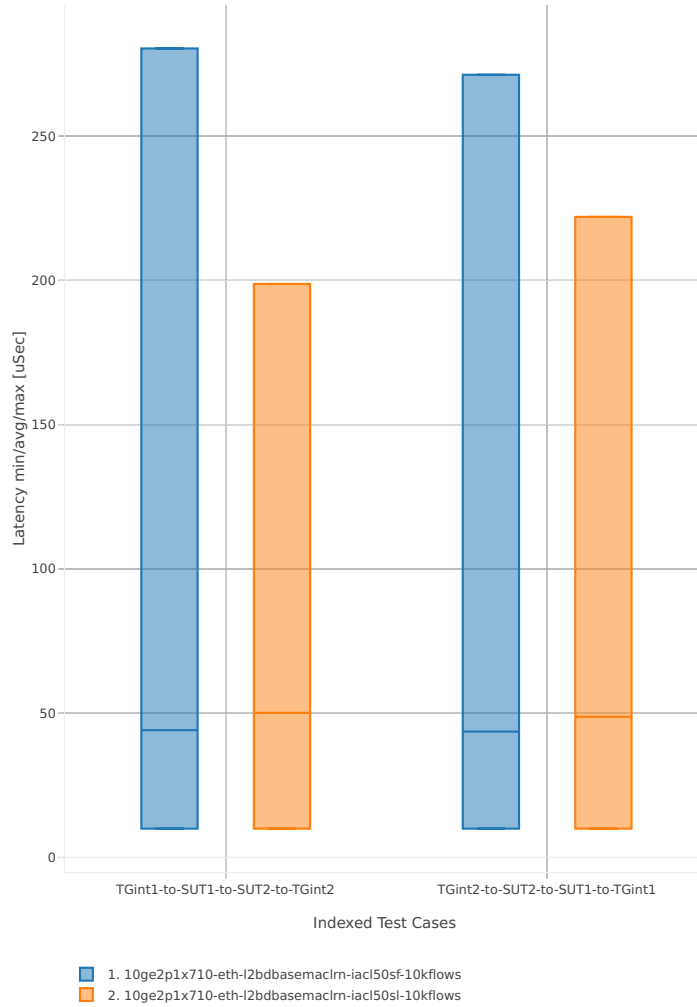
64b-2t1c-features

Latency: l2sw-3n-skx-x710-64b-2t1c-features-ndr



64b-4t2c-features

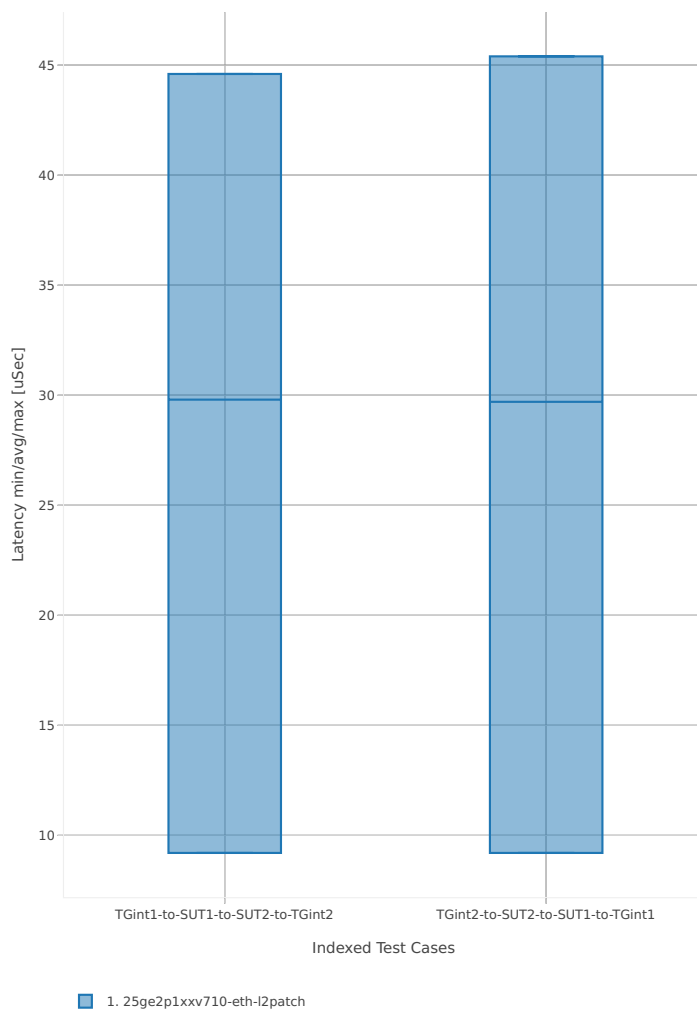
Latency: l2sw-3n-skx-x710-64b-4t2c-features-ndr



3n-skx-xxv710

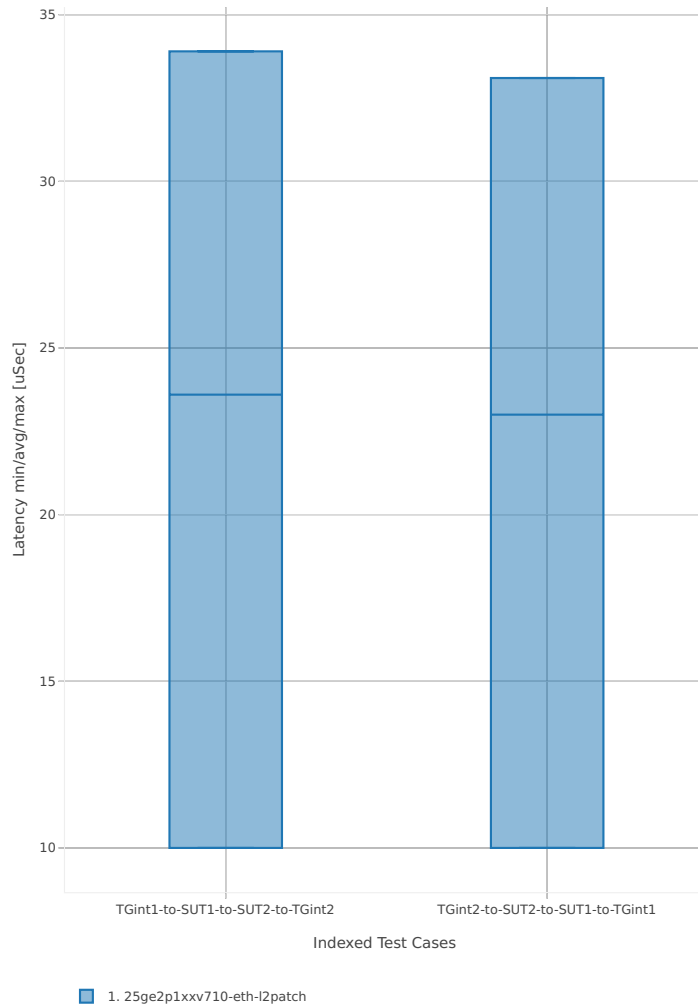
64b-2t1c-base_and_scale

Latency: l2sw-3n-skx-xxv710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

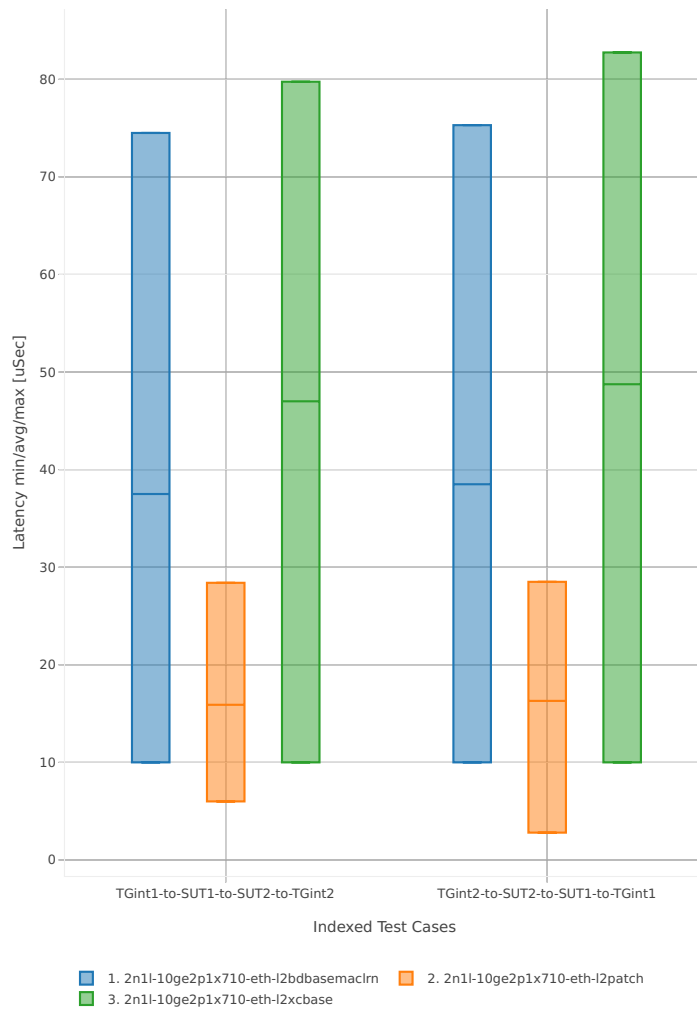
Latency: l2sw-3n-skx-xxv710-64b-4t2c-base_and_scale-ndr



2n-skx-x710

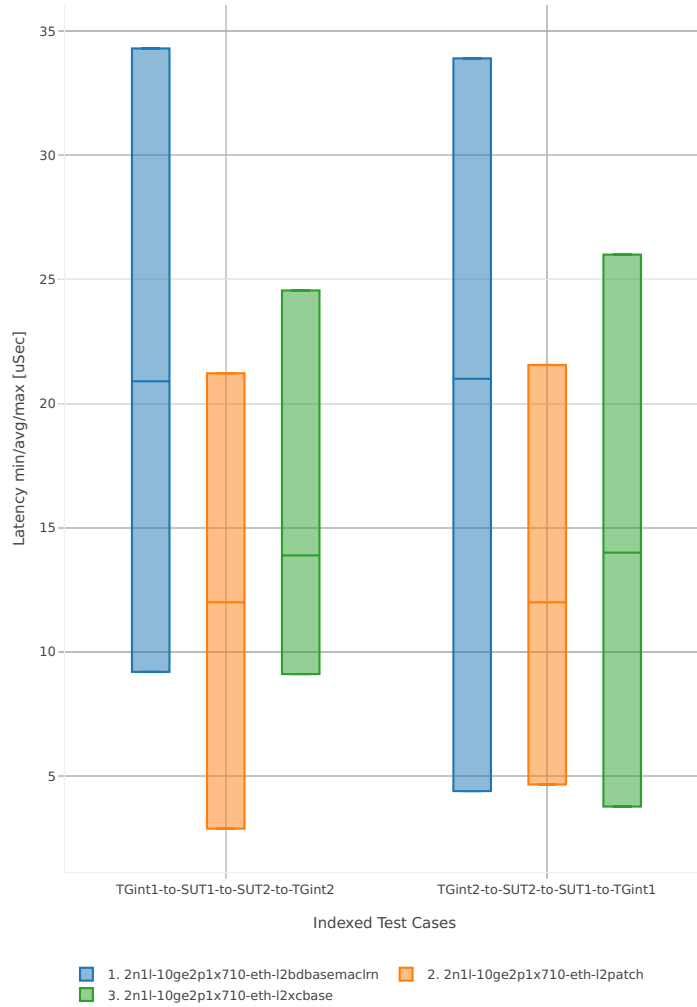
64b-2t1c-base_and_scale

Latency: l2sw-2n-skx-x710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

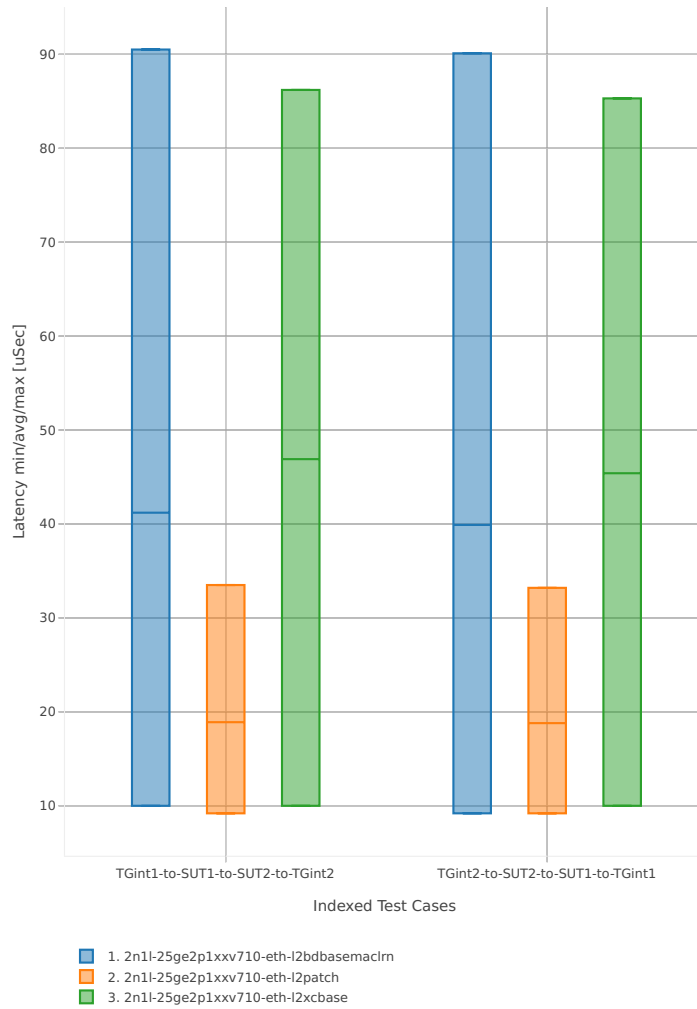
Latency: l2sw-2n-skx-x710-64b-4t2c-base_and_scale-ndr



2n-skx-xxv710

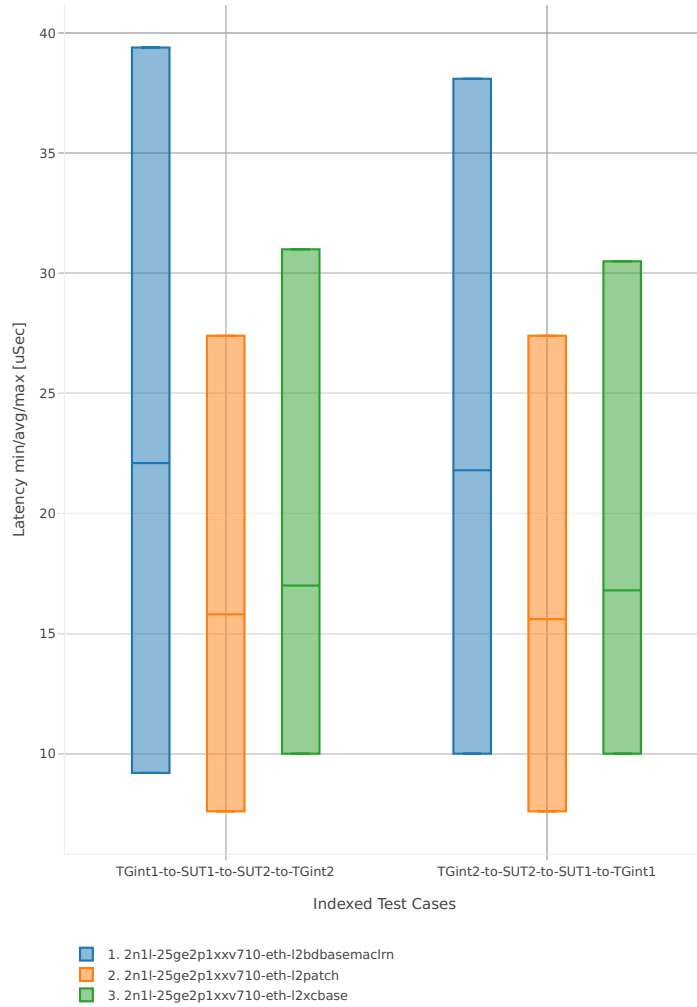
64b-2t1c-base_and_scale

Latency: l2sw-2n-skx-xxv710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

Latency: l2sw-2n-skx-xxv710-64b-4t2c-base_and_scale-ndr



2.5.2 IPv4 Routing

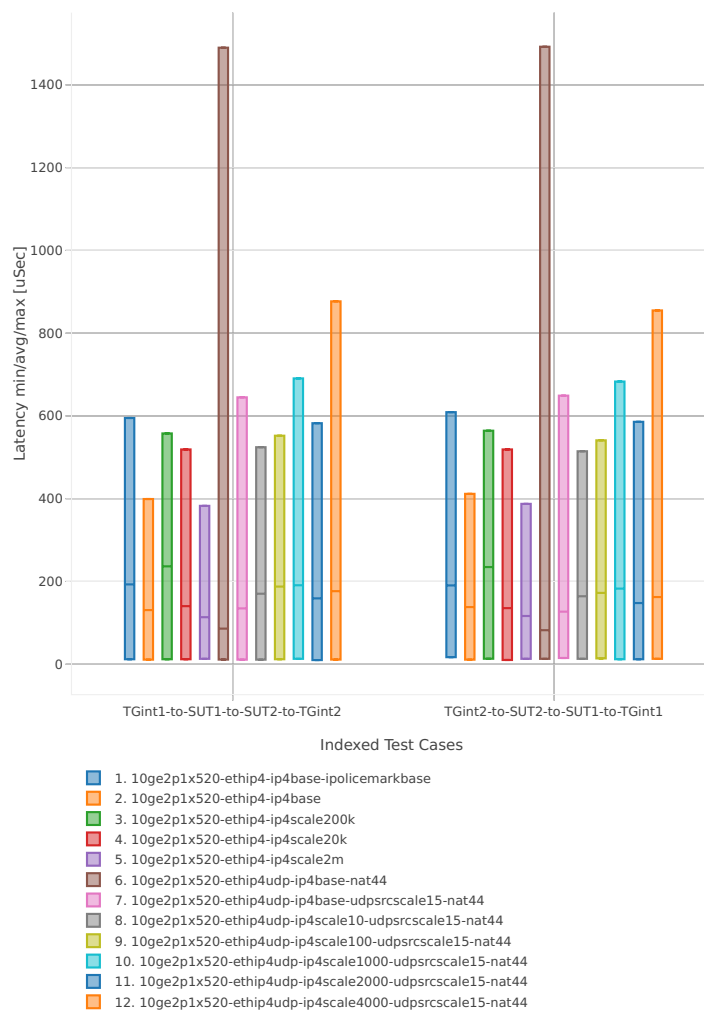
This section includes summary graphs of VPP Phy-to-Phy packet latency with IPv4 Routed-Forwarding measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁵⁹.

3n-hsw-x520

64b-1t1c-base_and_scale

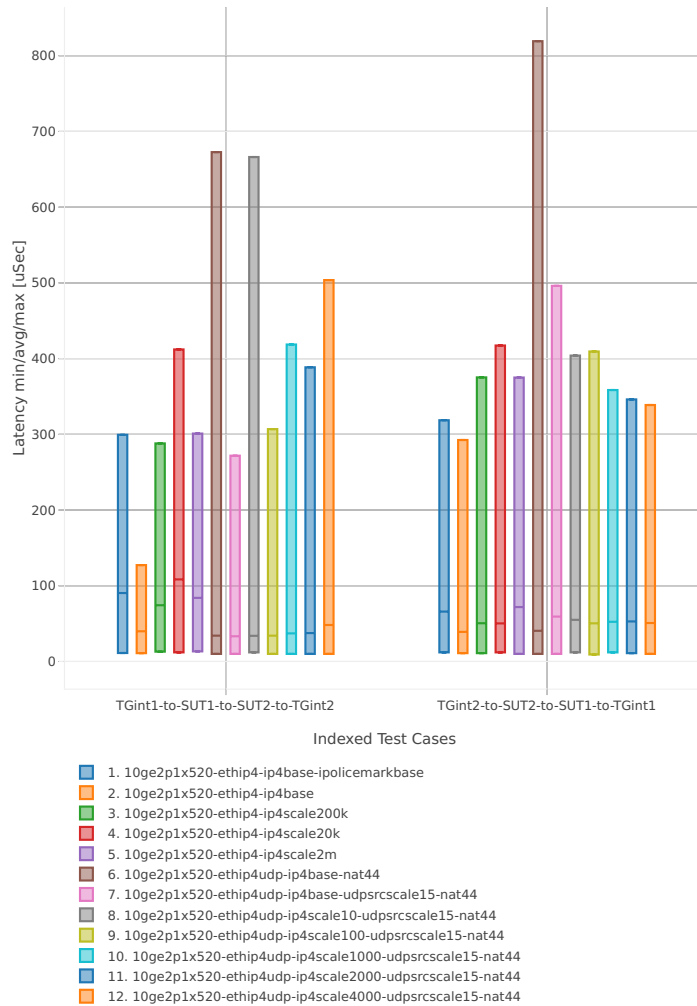
Latency: ip4-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁵⁹ <https://git.fd.io/csit/tree/tests/vpp/perf/ip4?h=rls1807>

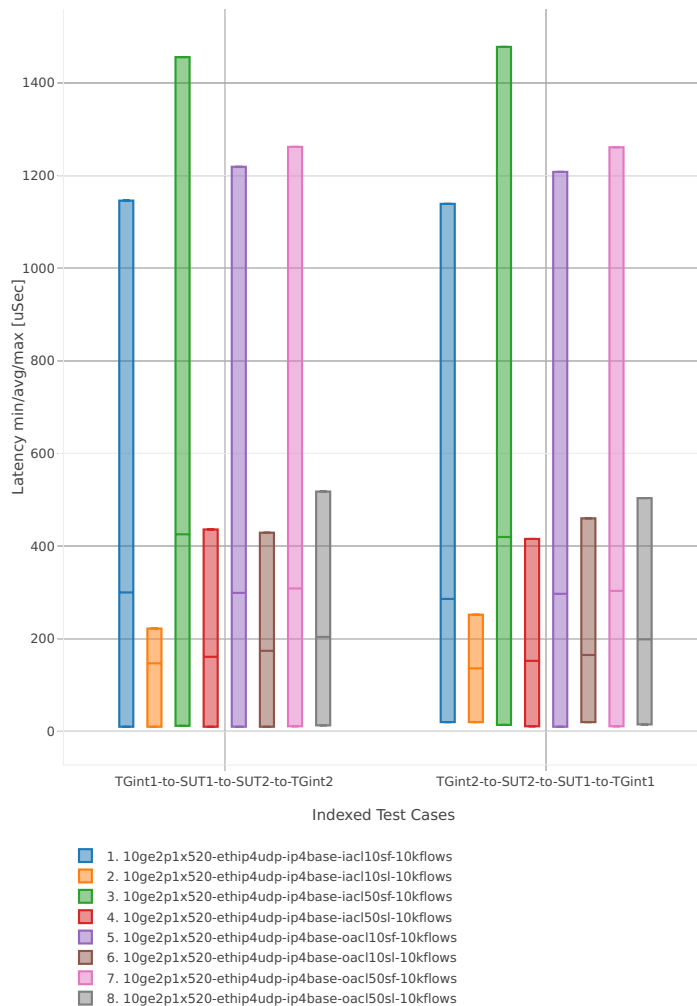
64b-2t2c-base_and_scale

Latency: ip4-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



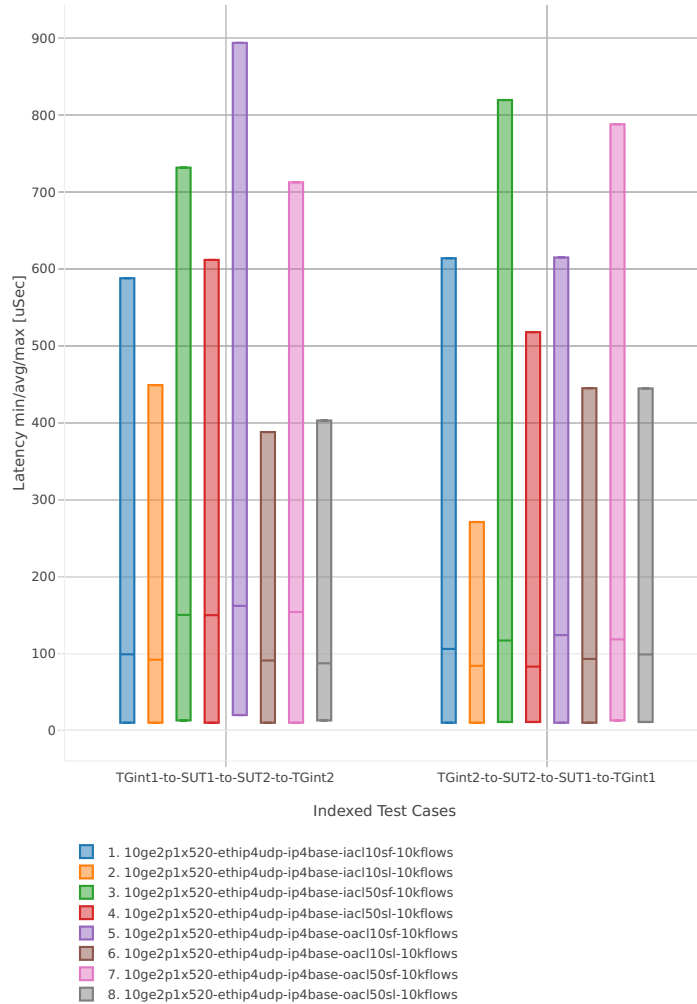
64b-1t1c-features

Latency: ip4-3n-hsw-x520-64b-1t1c-features-ndr



64b-2t2c-features

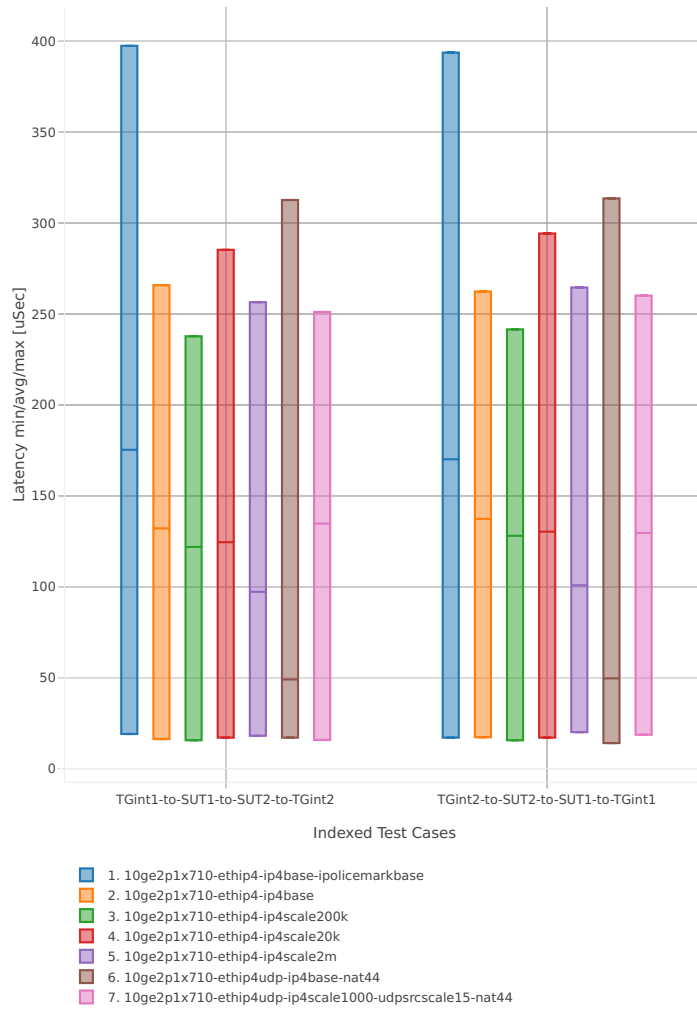
Latency: ip4-3n-hsw-x520-64b-2t2c-features-ndr



3n-hsw-x710

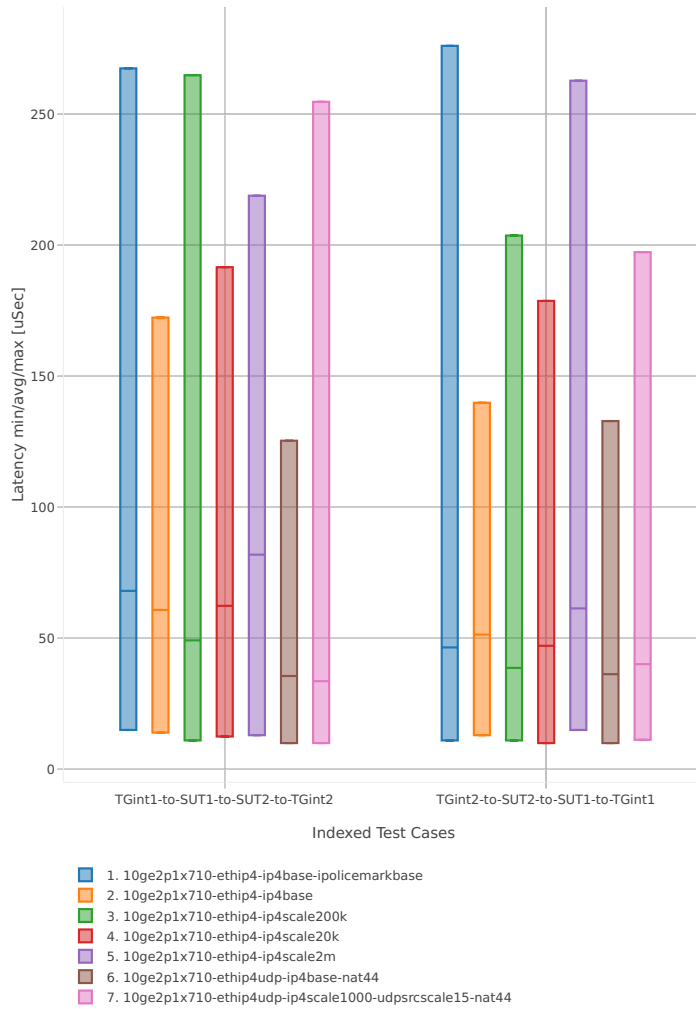
64b-1t1c-base_and_scale

Latency: ip4-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



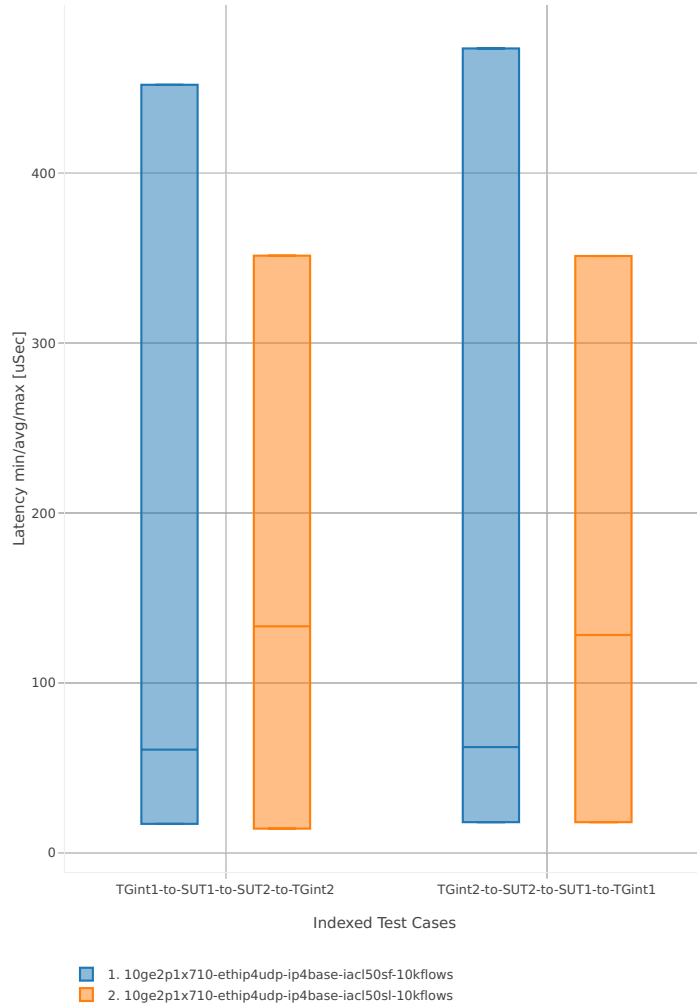
64b-2t2c-base_and_scale

Latency: ip4-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



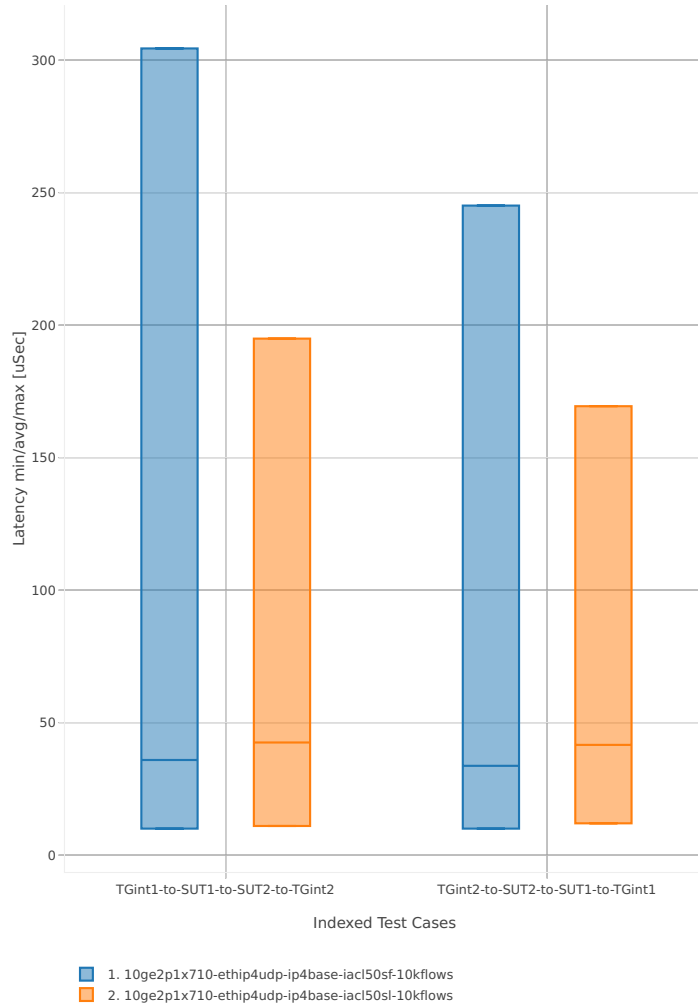
64b-1t1c-features

Latency: ip4-3n-hsw-x710-64b-1t1c-features-ndr



64b-2t2c-features

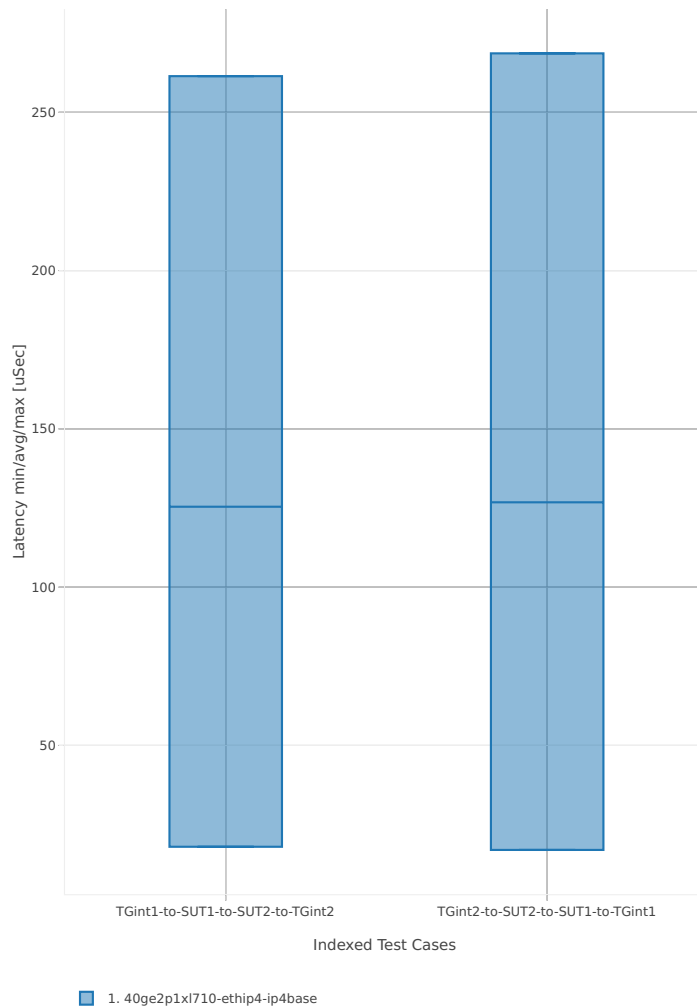
Latency: ip4-3n-hsw-x710-64b-2t2c-features-ndr



3n-hsw-xl710

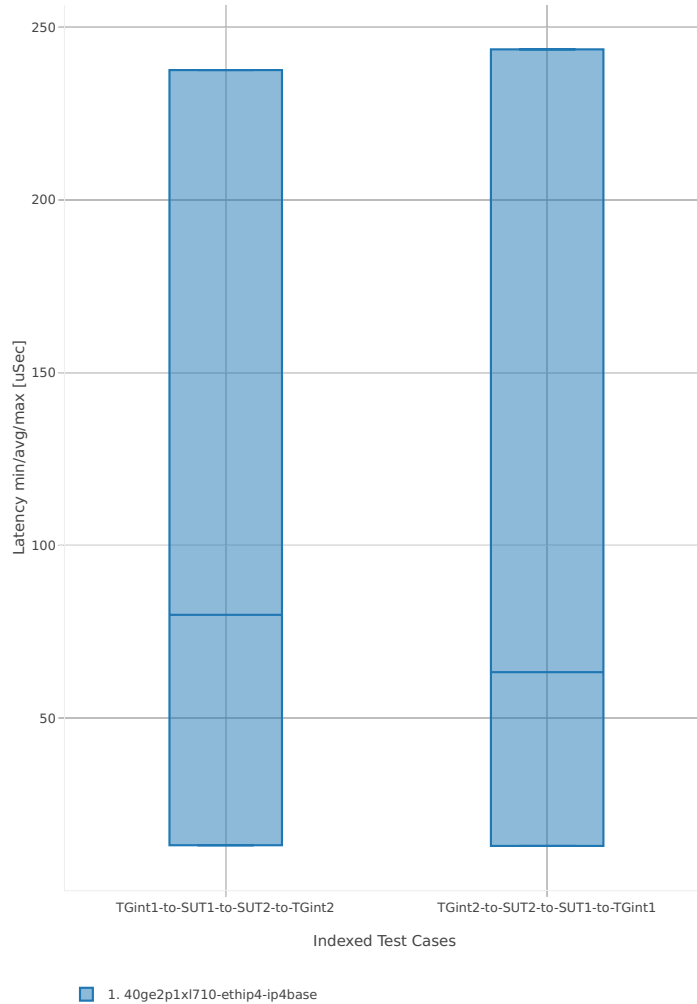
64b-1t1c-base_and_scale

Latency: ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

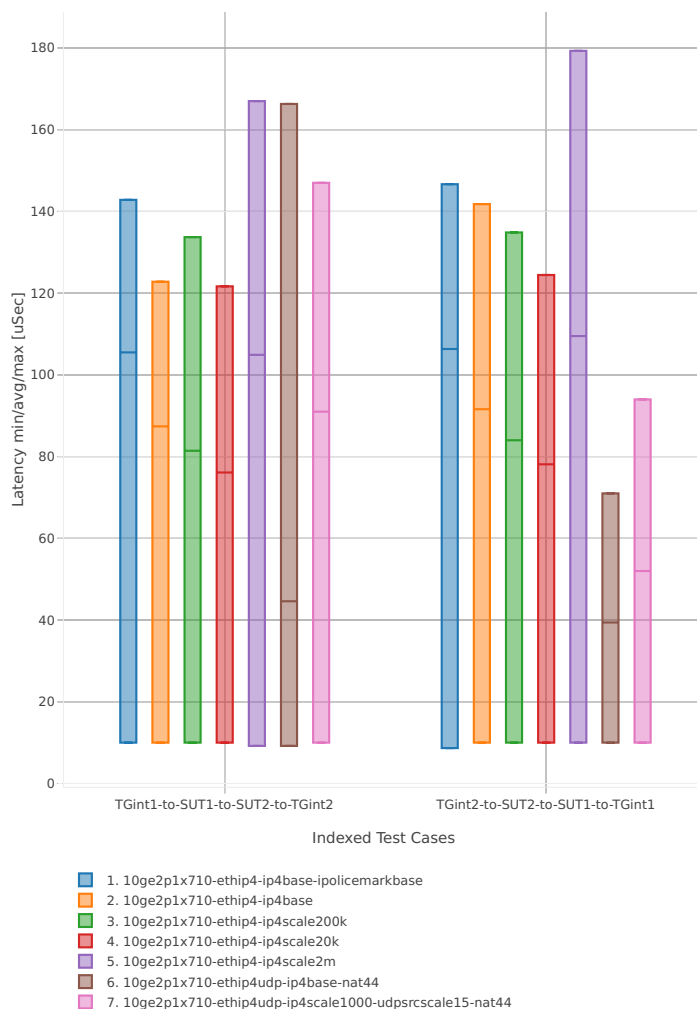
Latency: ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



3n-skx-x710

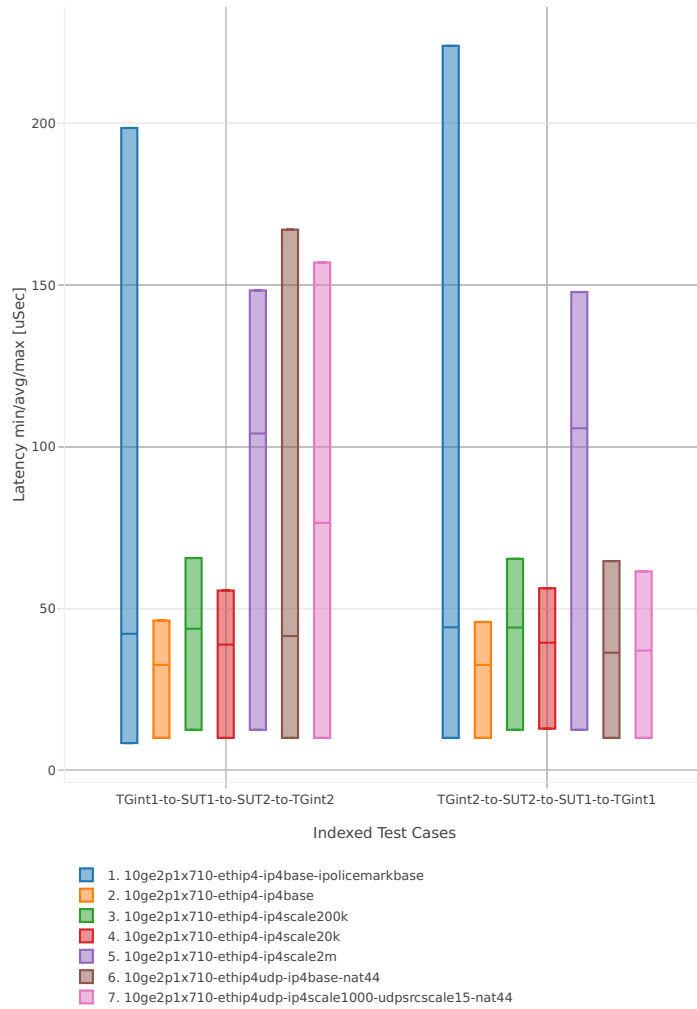
64b-2t1c-base_and_scale

Latency: ip4-3n-skx-x710-64b-2t1c-base_and_scale-ndr



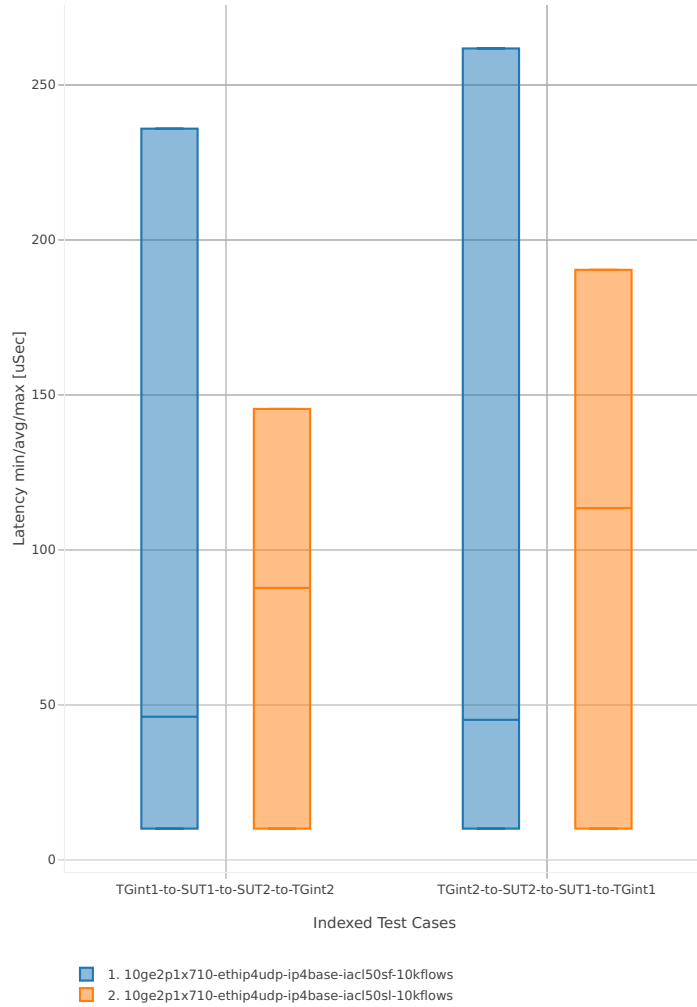
64b-4t2c-base_and_scale

Latency: ip4-3n-skx-x710-64b-4t2c-base_and_scale-ndr



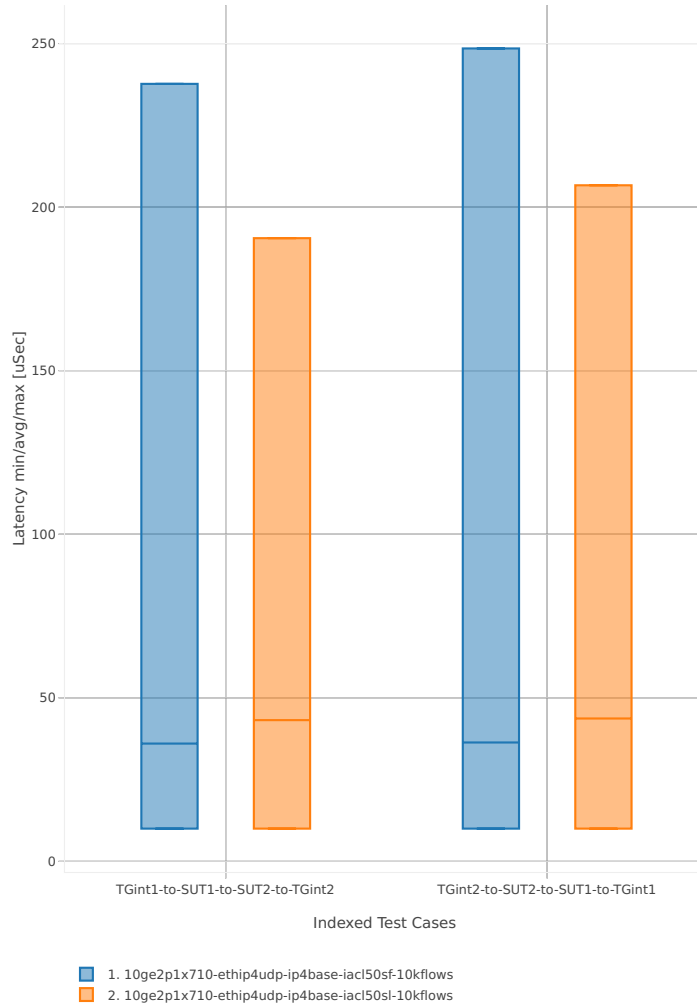
64b-2t1c-features

Latency: ip4-3n-skx-x710-64b-2t1c-features-ndr



64b-4t2c-features

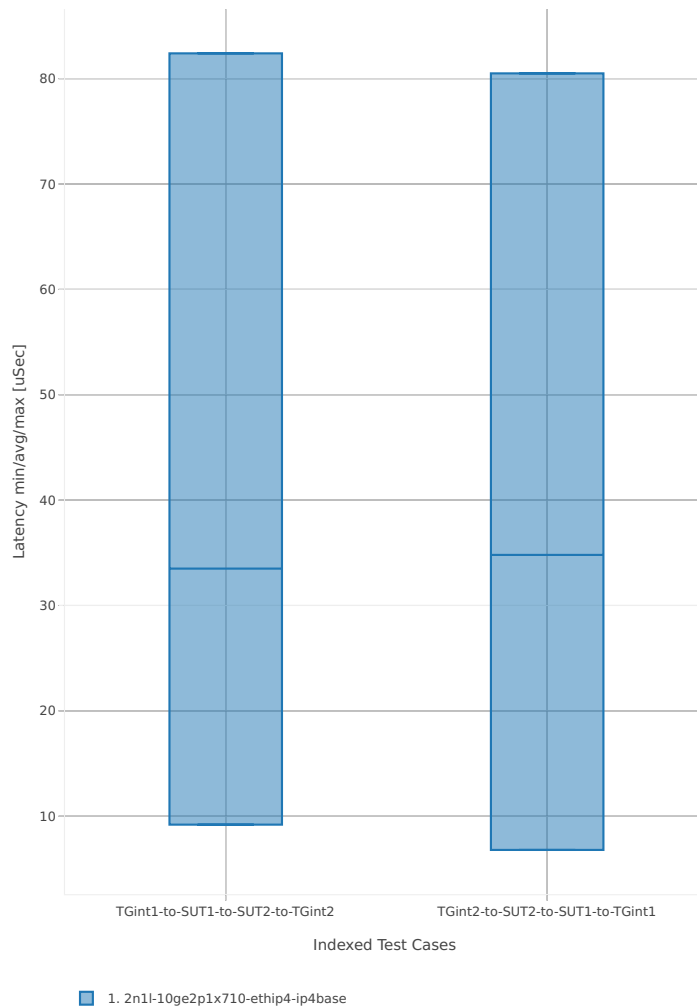
Latency: ip4-3n-skx-x710-64b-4t2c-features-ndr



2n-skx-x710

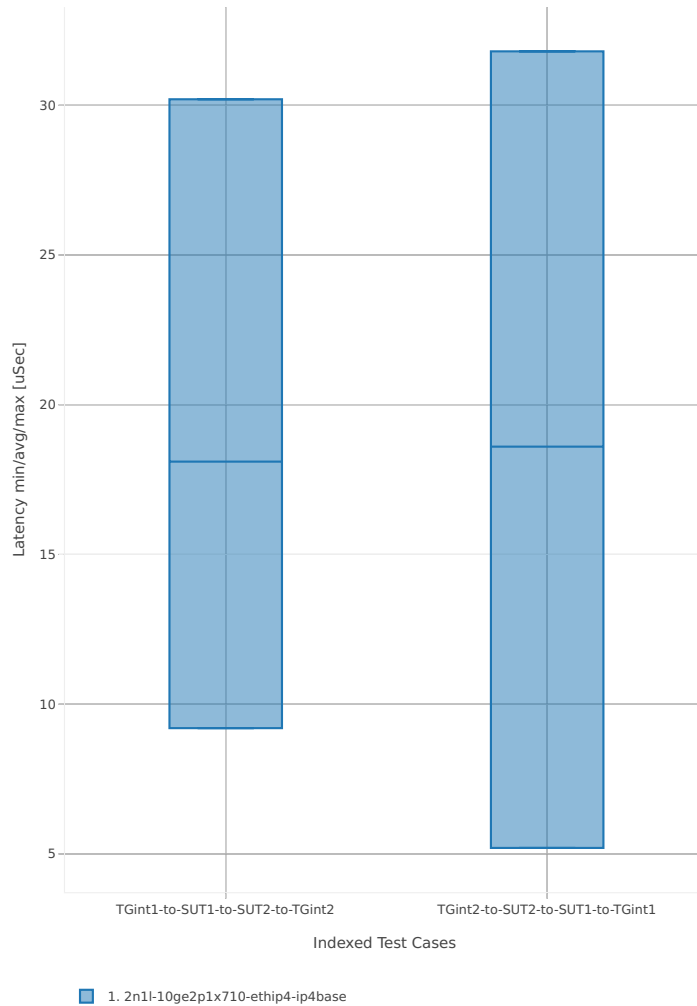
64b-2t1c-base_and_scale

Latency: ip4-2n-skx-x710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

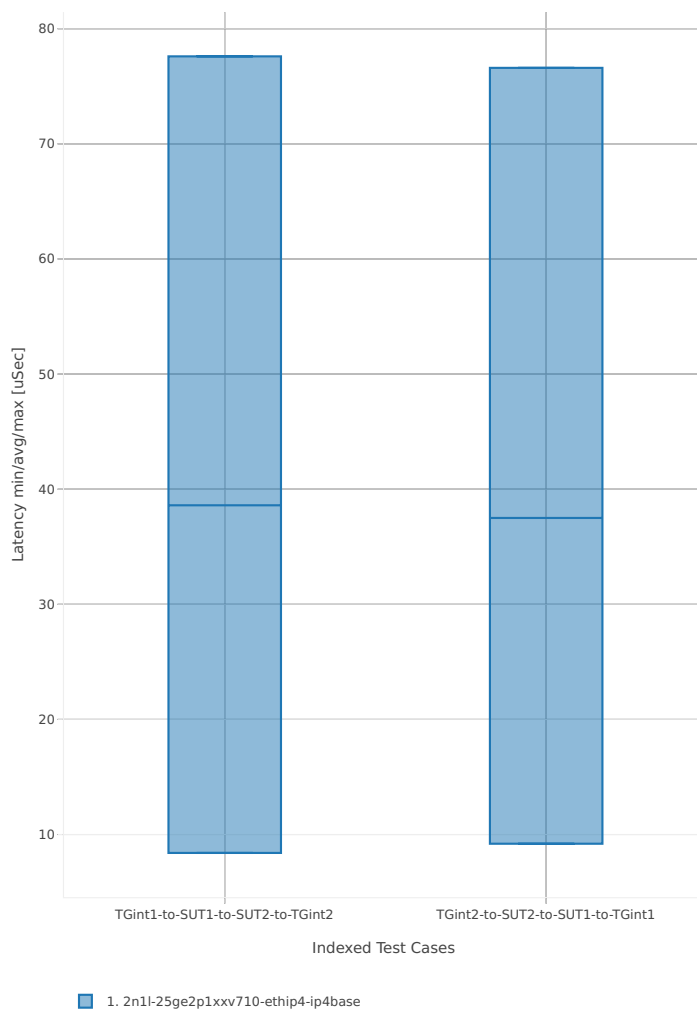
Latency: ip4-2n-skx-x710-64b-4t2c-base_and_scale-ndr



2n-skx-xxv710

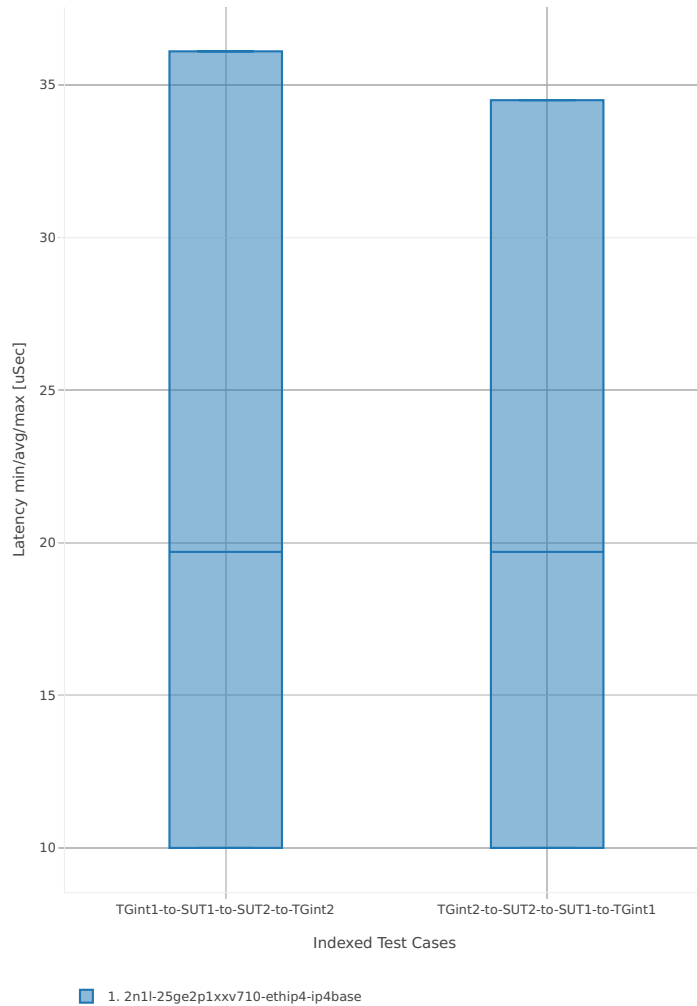
64b-2t1c-base_and_scale

Latency: ip4-2n-skx-xxv710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

Latency: ip4-2n-skx-xxv710-64b-4t2c-base_and_scale-ndr



2.5.3 IPv6 Routing

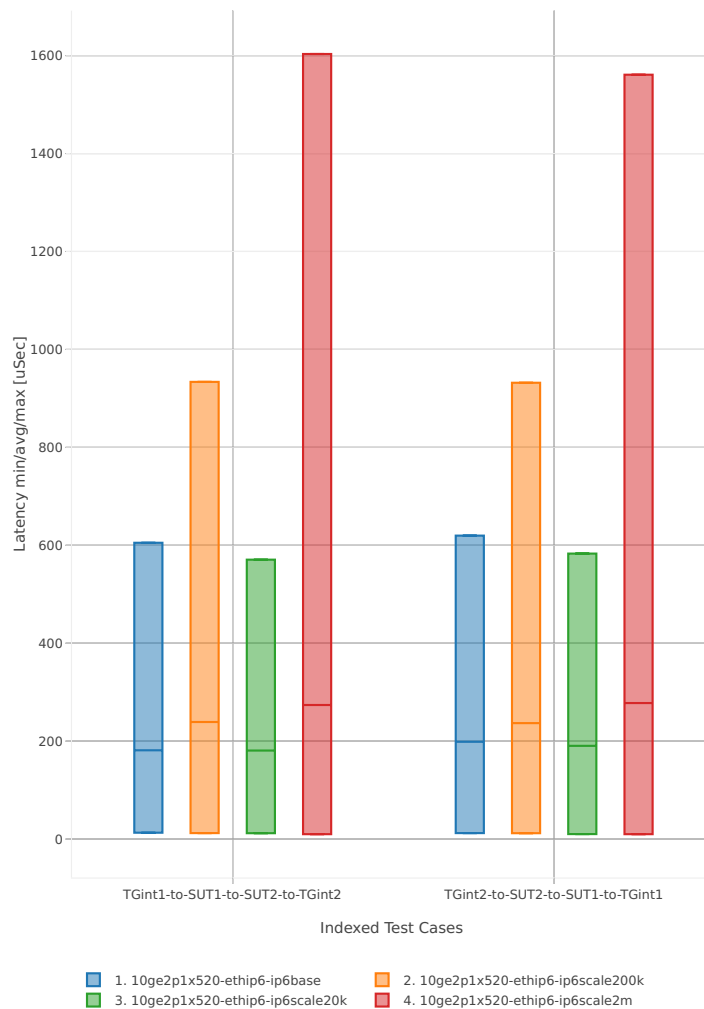
This section includes summary graphs of VPP Phy-to-Phy packet latency with IPv6 Routed-Forwarding measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶⁰.

3n-hsw-x520

78b-1t1c-base_and_scale

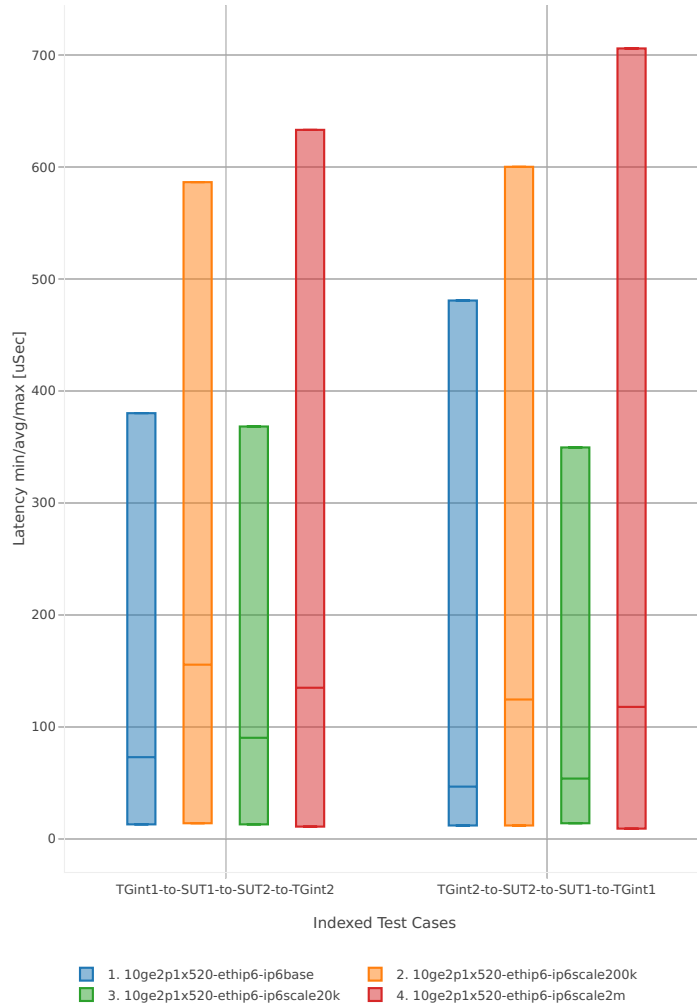
Latency: ip6-3n-hsw-x520-78b-1t1c-base_and_scale-ndr



⁶⁰ <https://git.fd.io/csit/tree/tests/vpp/perf/ip6?h=rls1807>

78b-2t2c-base_and_scale

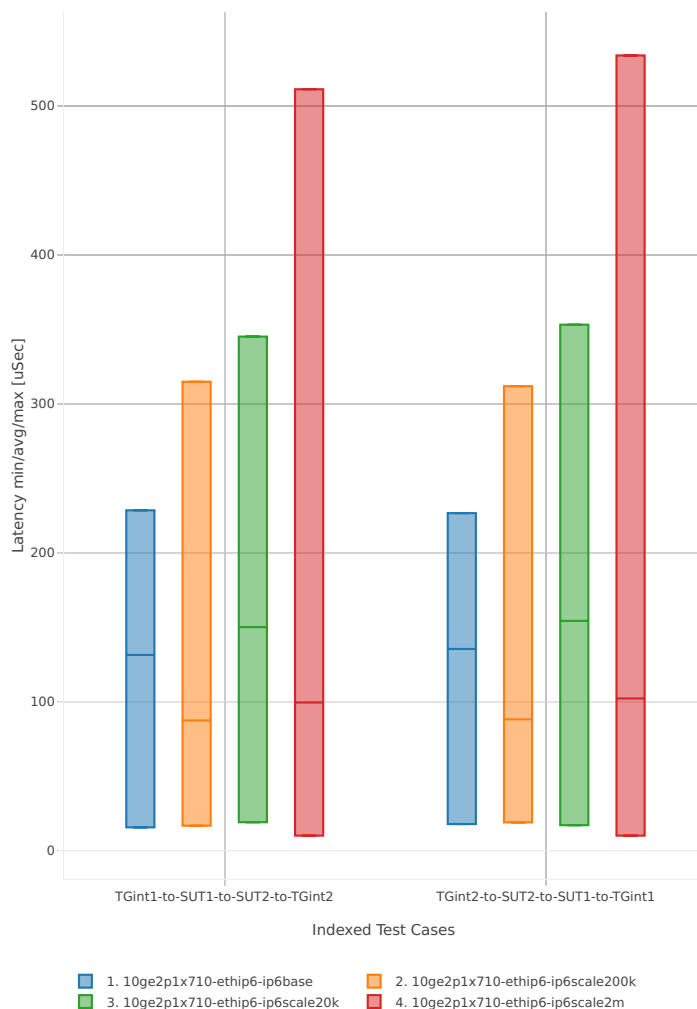
Latency: ip6-3n-hsw-x520-78b-2t2c-base_and_scale-ndr



3n-hsw-x710

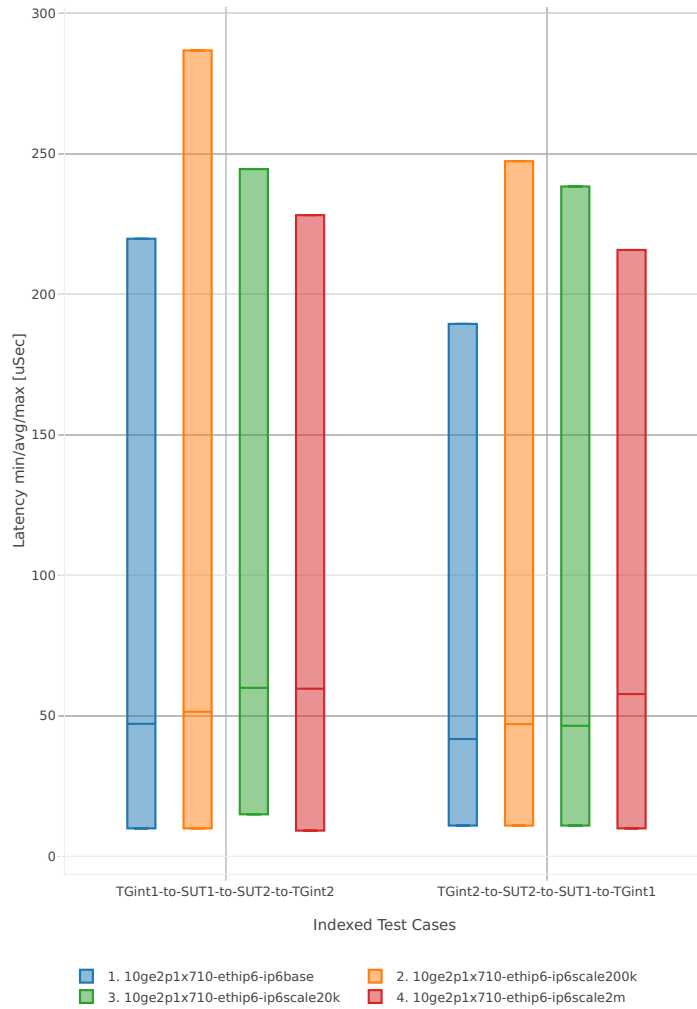
78b-1t1c-base_and_scale

Latency: ip6-3n-hsw-x710-78b-1t1c-base_and_scale-ndr



78b-2t2c-base_and_scale

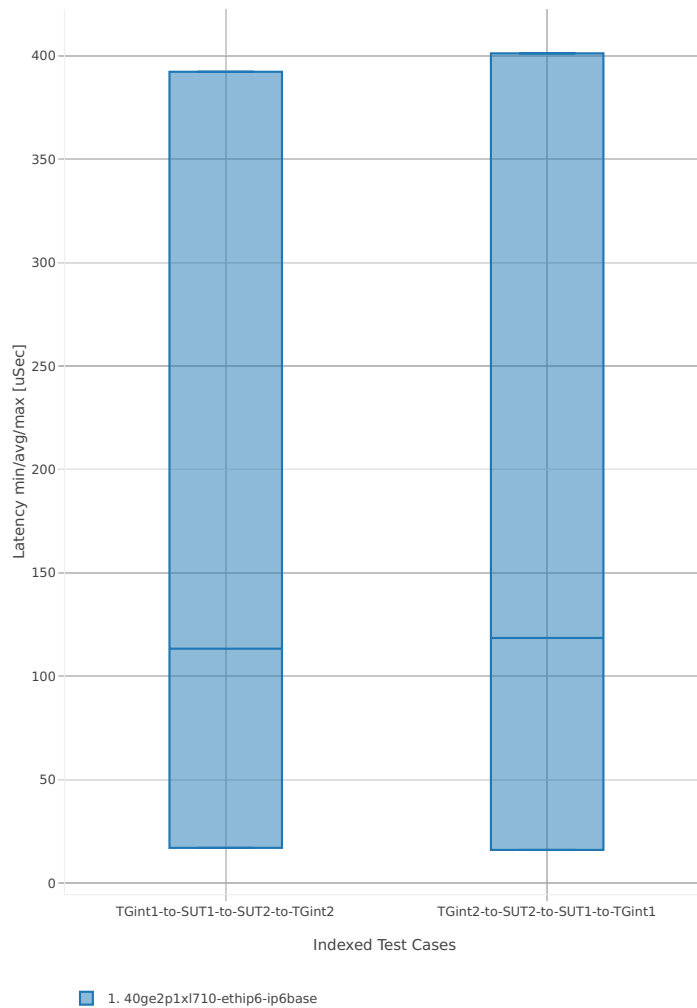
Latency: ip6-3n-hsw-x710-78b-2t2c-base_and_scale-ndr



3n-hsw-xl710

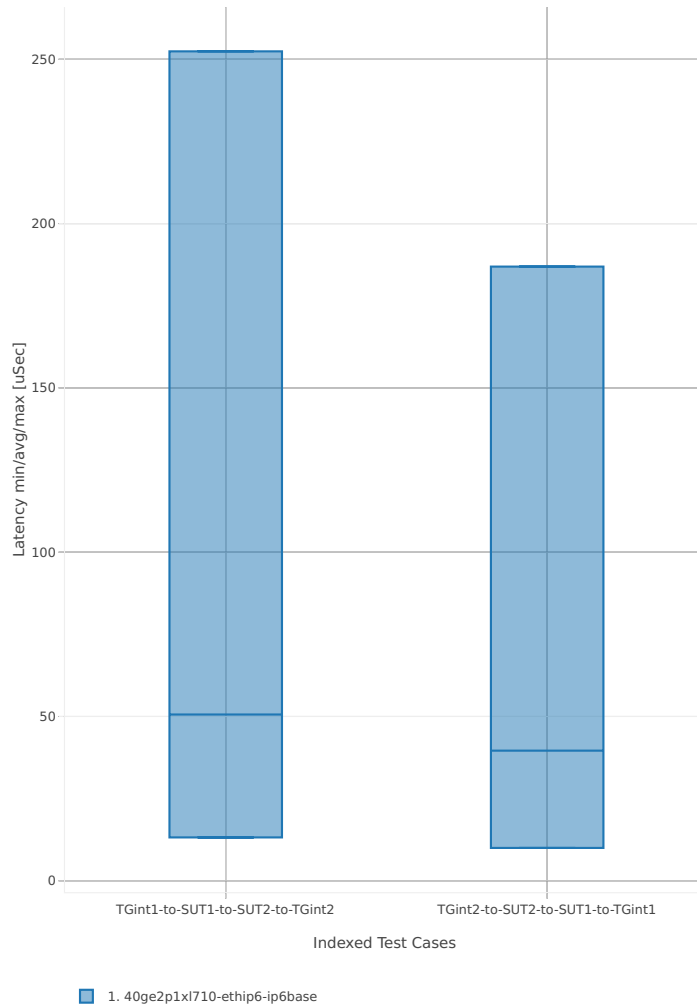
78b-1t1c-base_and_scale

Latency: ip6-3n-hsw-xl710-78b-1t1c-base_and_scale-ndr



78b-2t2c-base_and_scale

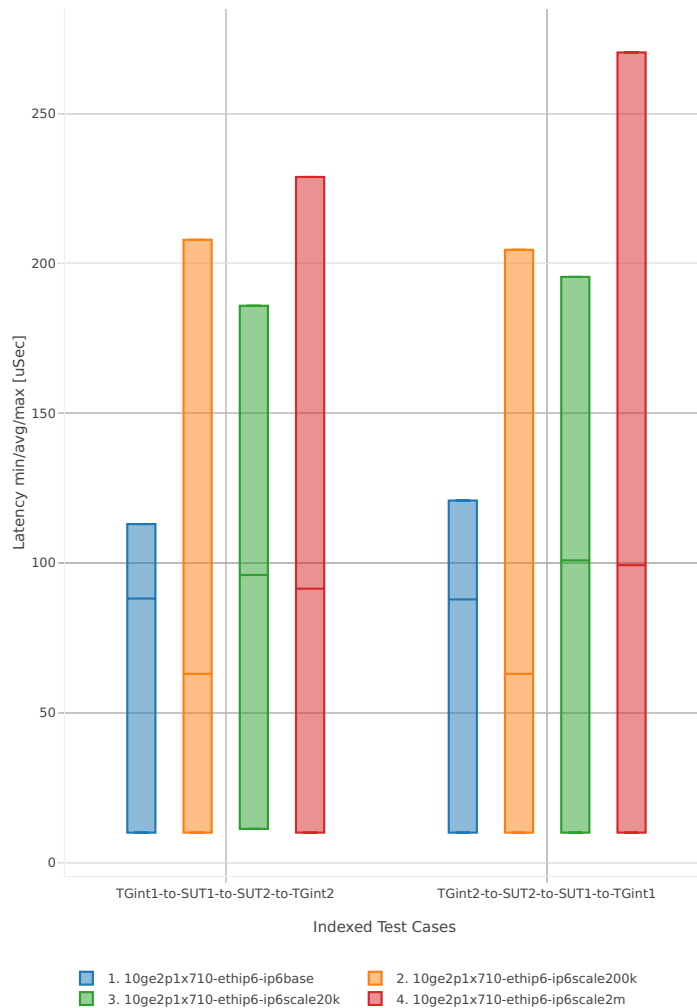
Latency: ip6-3n-hsw-xl710-78b-2t2c-base_and_scale-ndr



3n-skx-x710

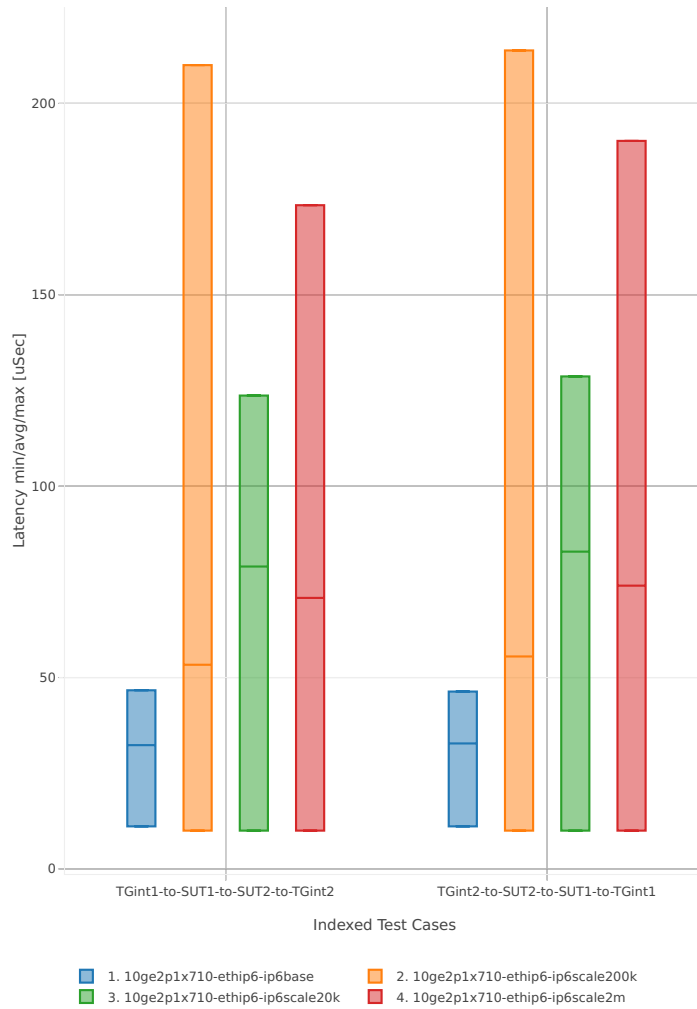
78b-2t1c-base_and_scale

Latency: ip6-3n-skx-x710-78b-2t1c-base_and_scale-ndr



78b-4t2c-base_and_scale

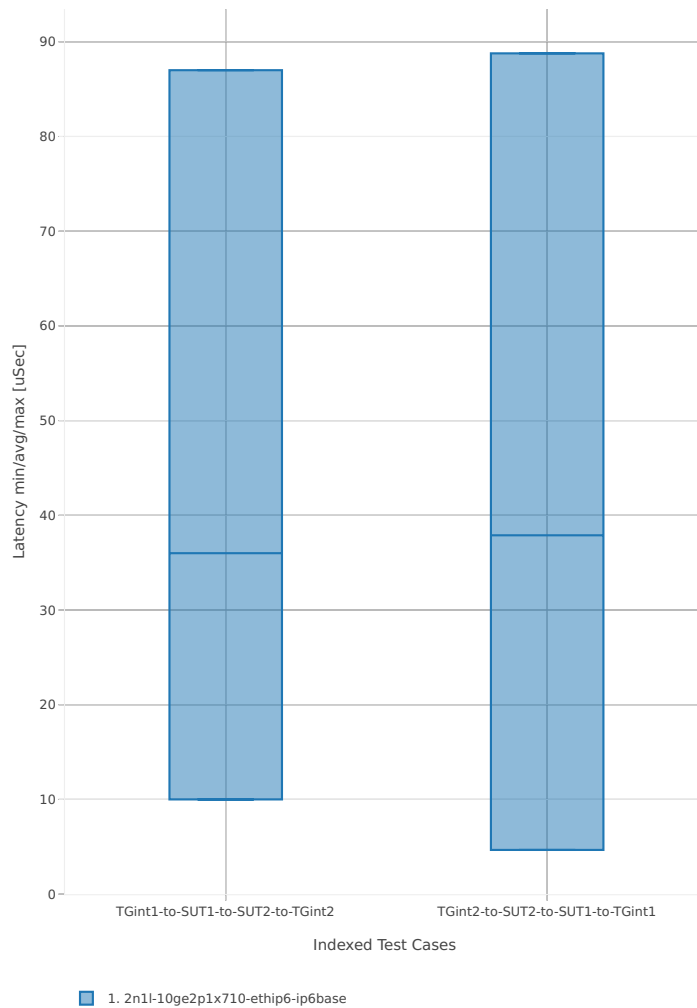
Latency: ip6-3n-skx-x710-78b-4t2c-base_and_scale-ndr



2n-skx-x710

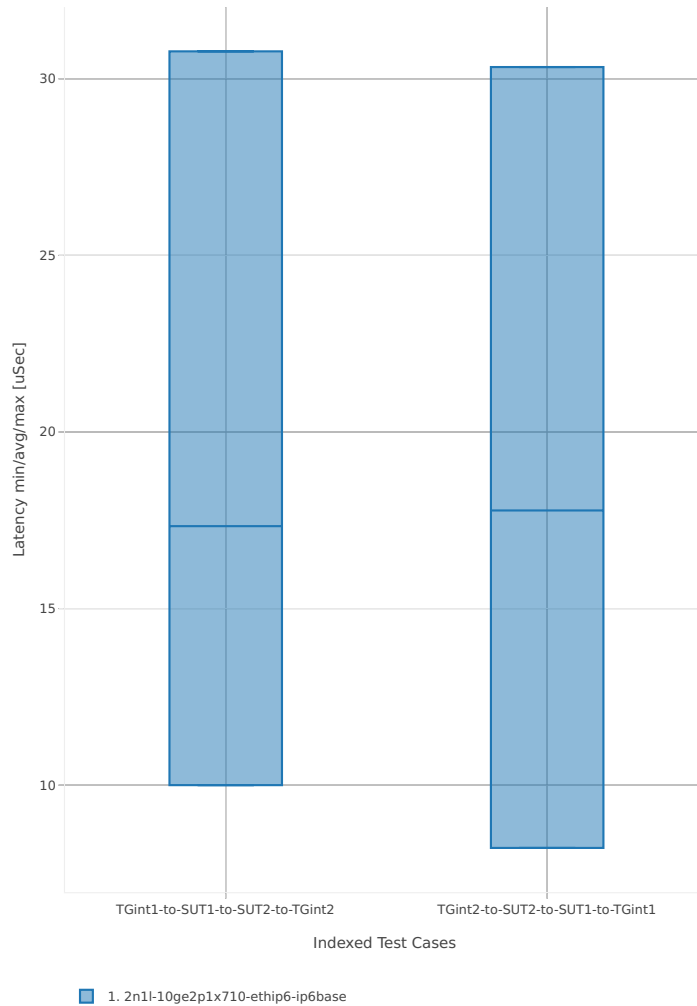
78b-2t1c-base_and_scale

Latency: ip6-2n-skx-x710-78b-2t1c-base_and_scale-ndr



78b-4t2c-base_and_scale

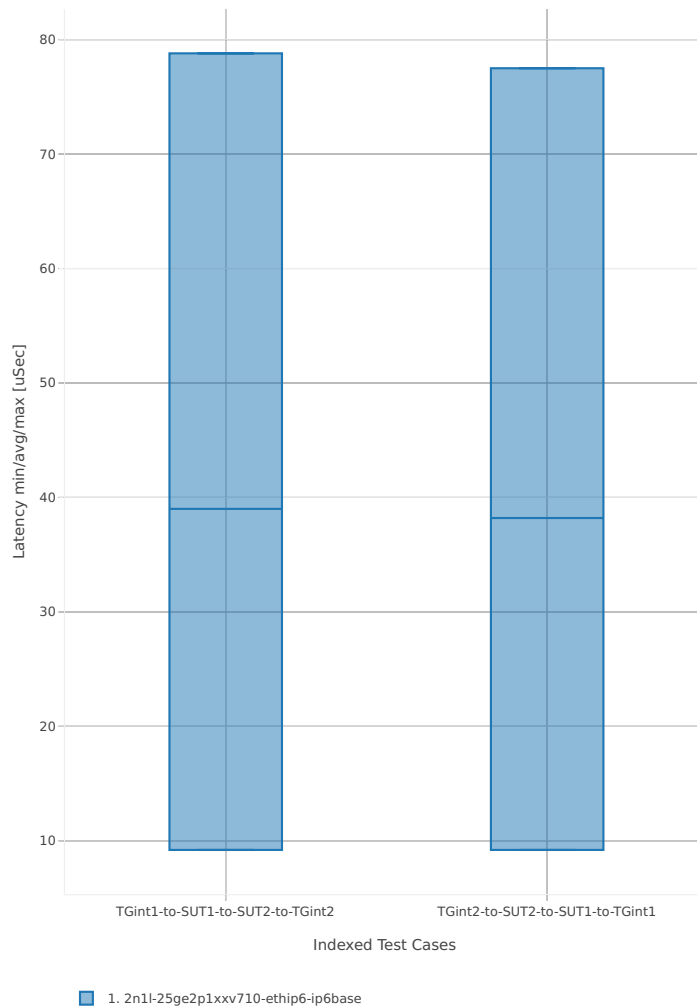
Latency: ip6-2n-skx-x710-78b-4t2c-base_and_scale-ndr



2n-skx-xxv710

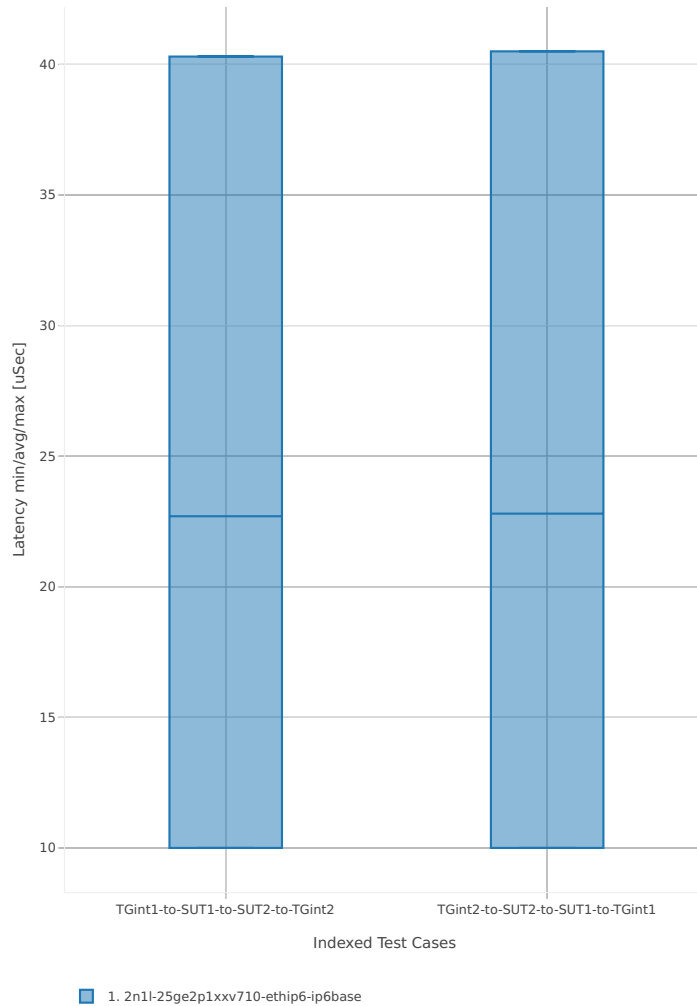
78b-2t1c-base_and_scale

Latency: ip6-2n-skx-xxv710-78b-2t1c-base_and_scale-ndr



78b-4t2c-base_and_scale

Latency: ip6-2n-skx-xxv710-78b-4t2c-base_and_scale-ndr



2.5.4 SRv6 Routing

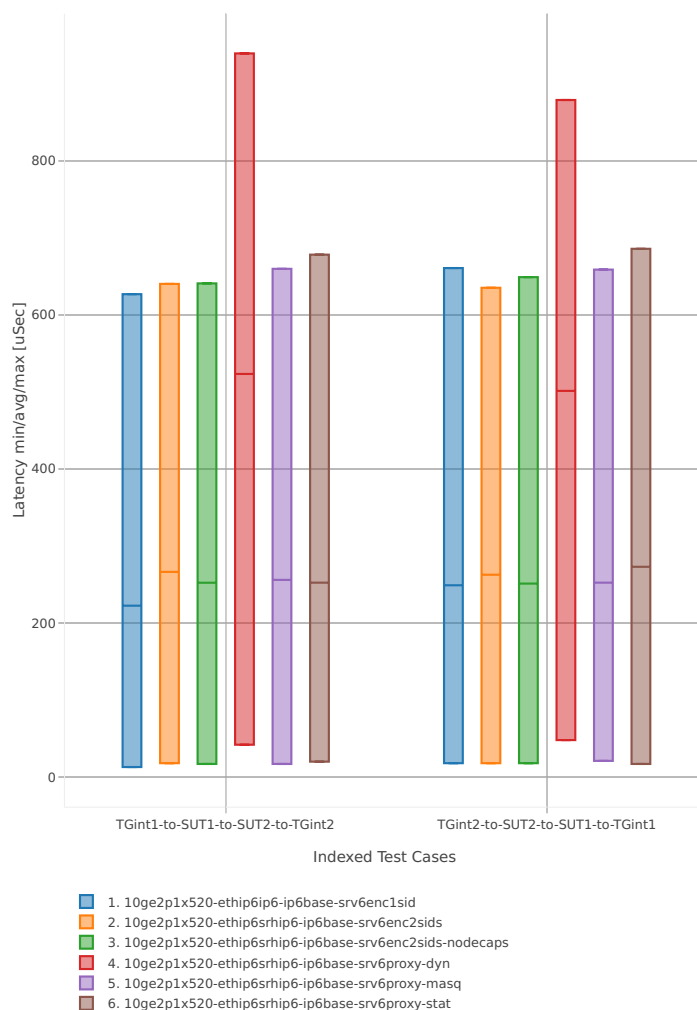
This section includes summary graphs of VPP Phy-to-Phy packet latency with SRv6 measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶¹.

3n-hsw-x520

78b-1t1c-features

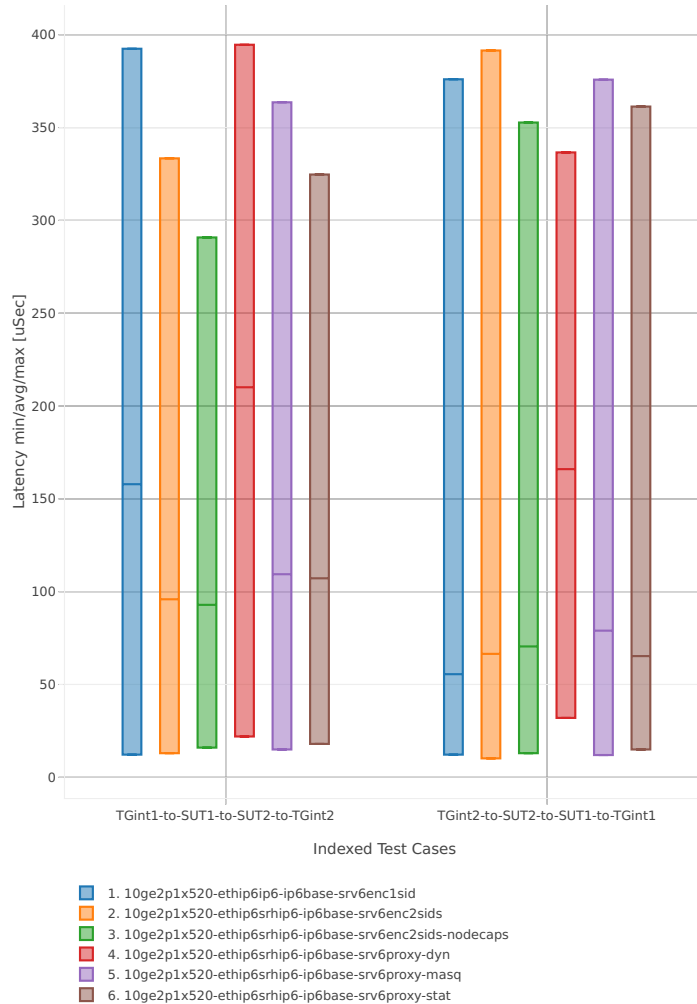
Latency: srv6-3n-hsw-x520-78b-1t1c-features-ndr



⁶¹ <https://git.fd.io/csit/tree/tests/vpp/perf/srv6?h=rls1807>

78b-2t2c-features

Latency: srv6-3n-hsw-x520-78b-2t2c-features-ndr



2.5.5 IPv4 Tunnels

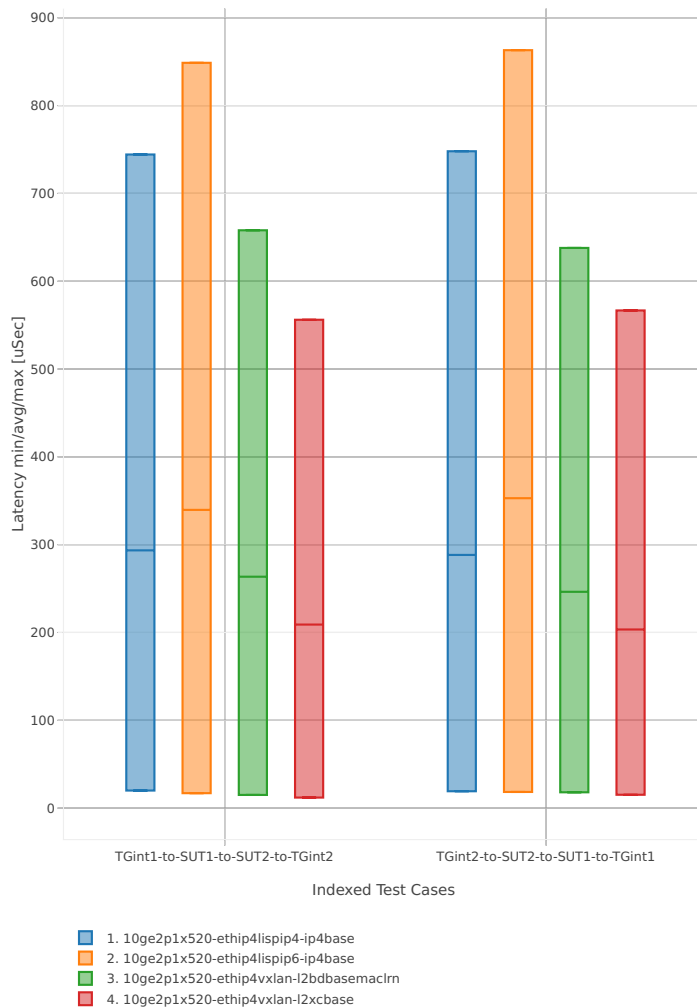
This section includes summary graphs of VPP Phy-to-Phy packet latency with IPv4 Overlay Tunnels measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶².

3n-hsw-x520

64b-1t1c-base_and_scale

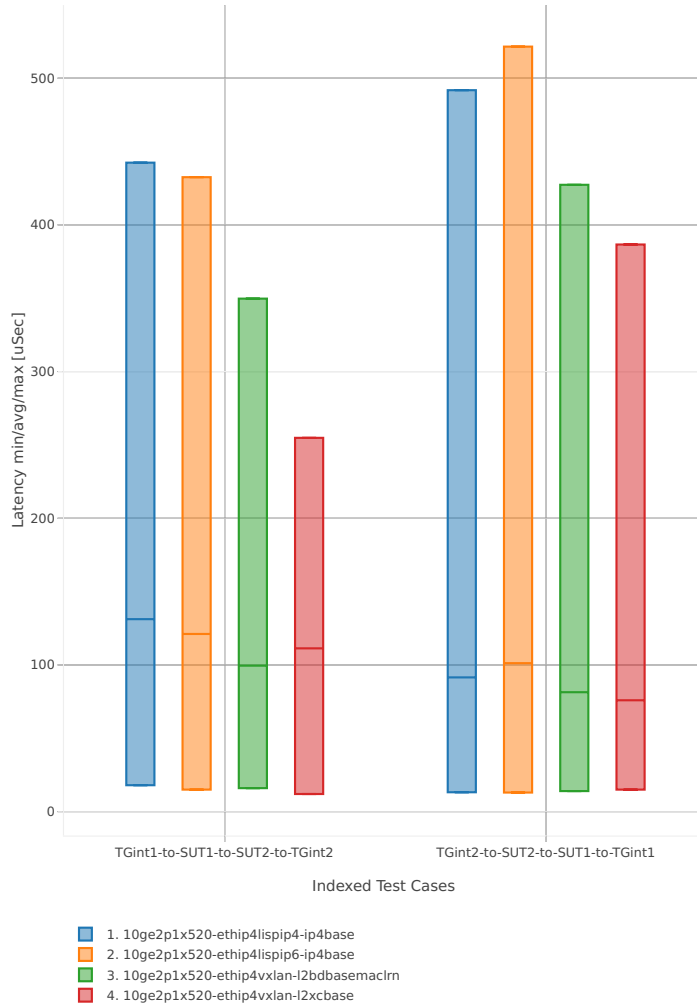
Latency: ip4tun-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁶² https://git.fd.io/csit/tree/tests/vpp/perf/ip4_tunnels?h=rls1807

64b-2t2c-base_and_scale

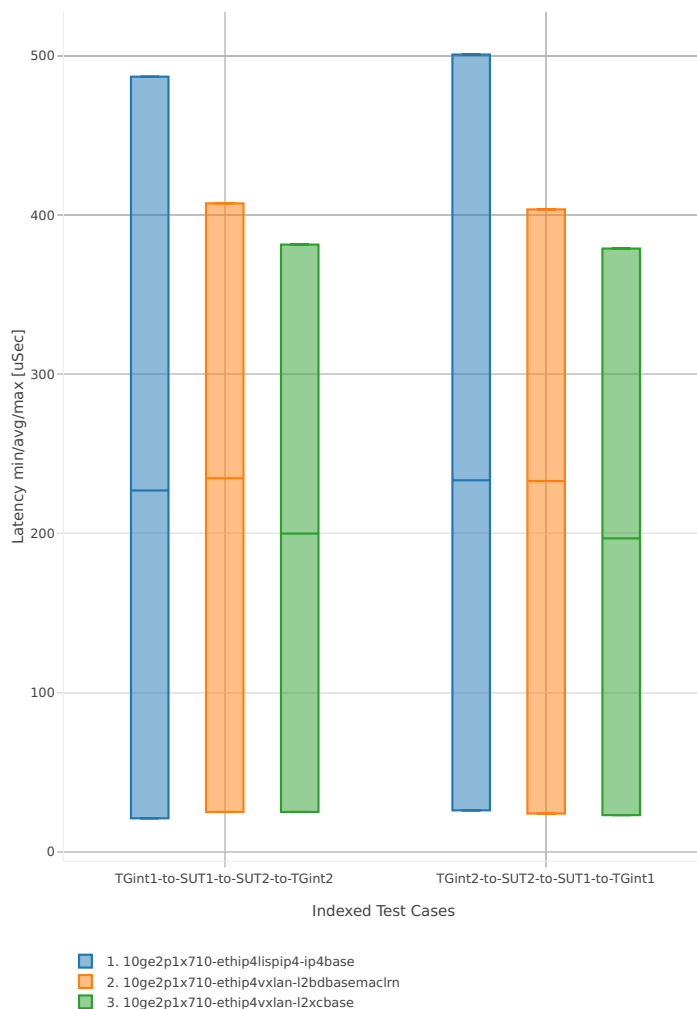
Latency: ip4tun-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



3n-hsw-x710

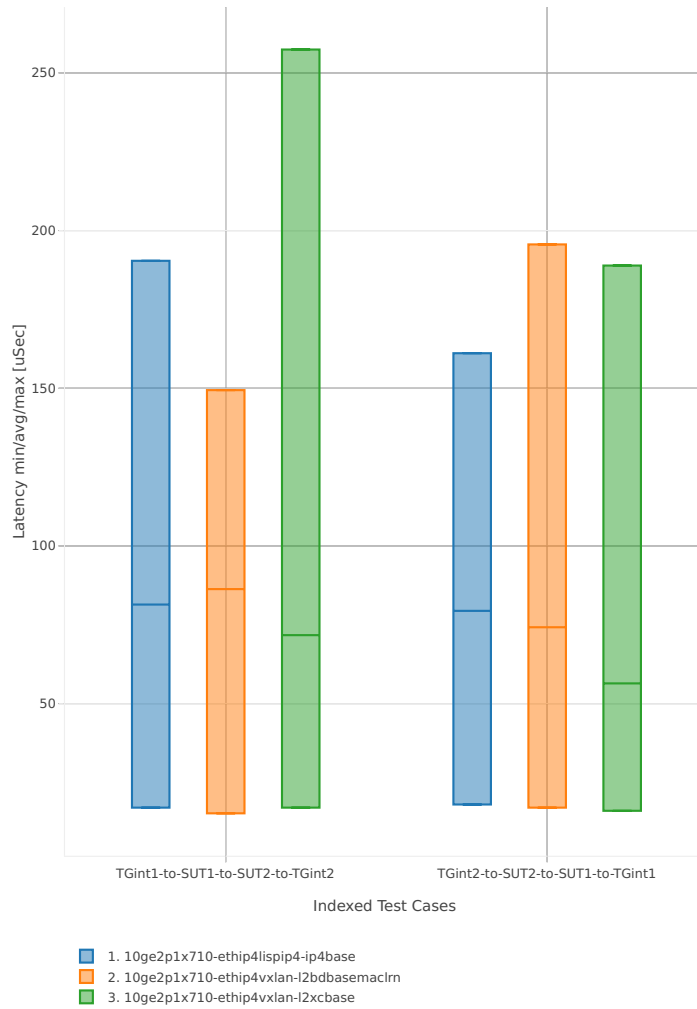
64b-1t1c-base_and_scale

Latency: ip4tun-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

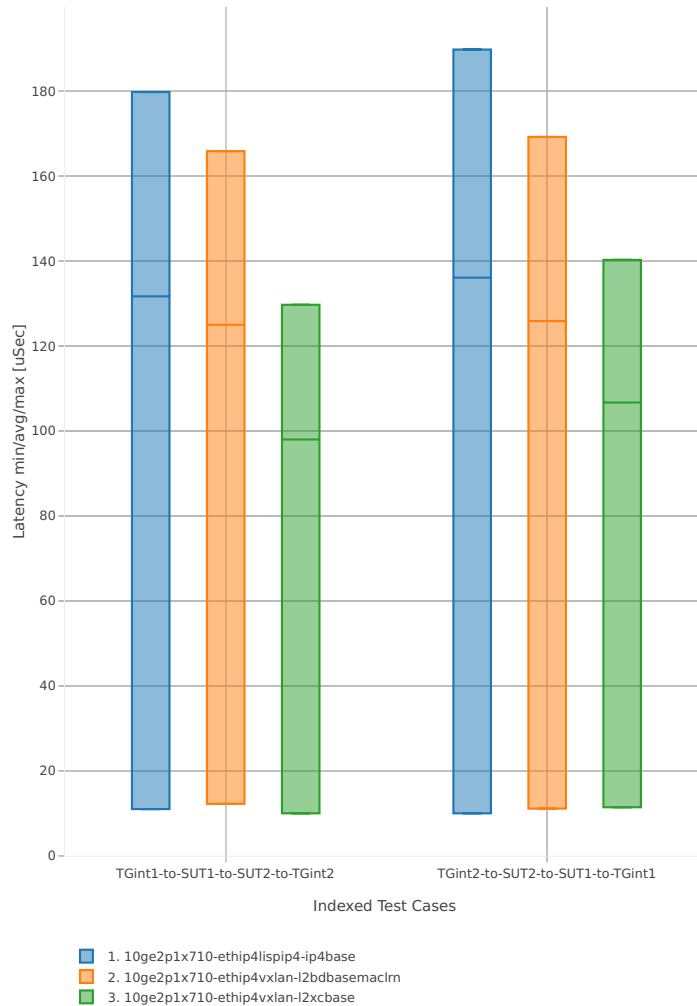
Latency: ip4tun-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



3n-skx-x710

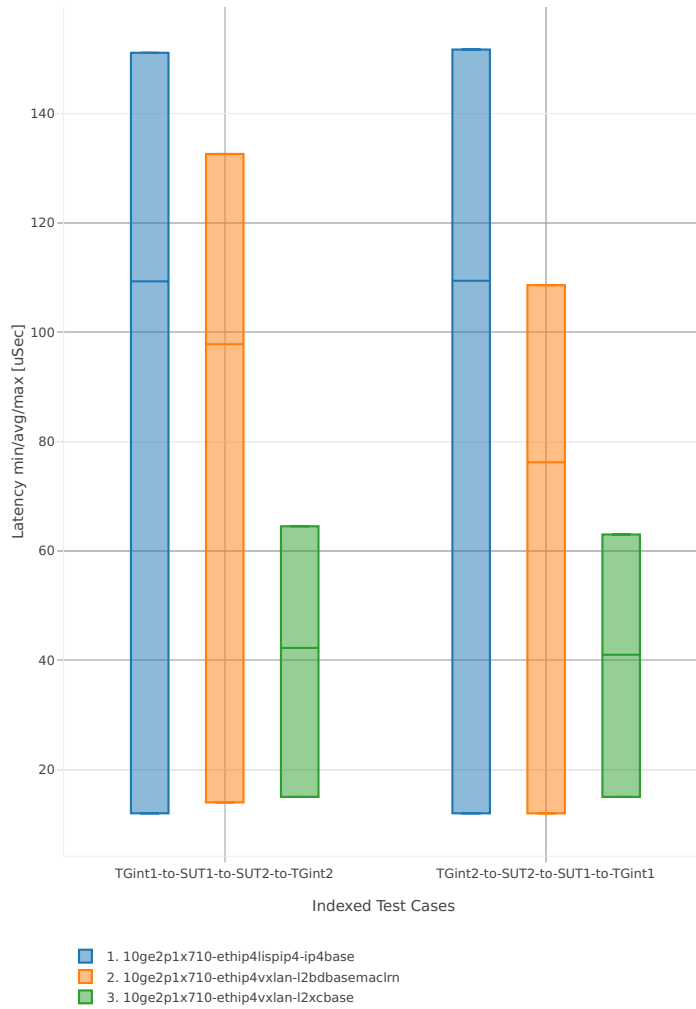
64b-2t1c-base_and_scale

Latency: ip4tun-3n-skx-x710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

Latency: ip4tun-3n-skx-x710-64b-4t2c-base_and_scale-ndr



2.5.6 IPv6 Tunnels

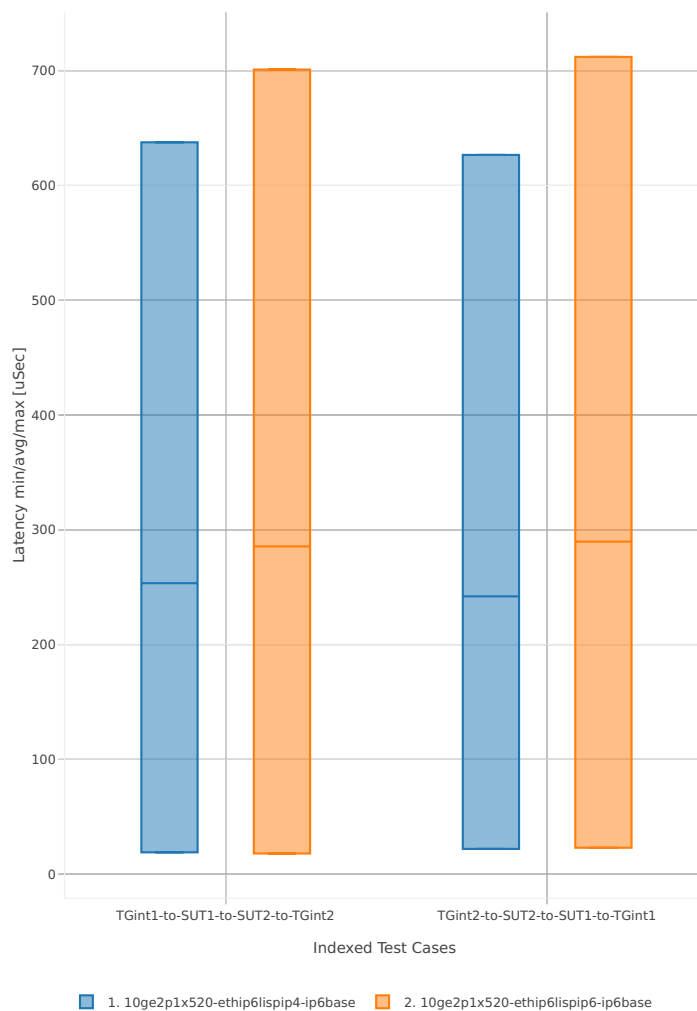
This section includes summary graphs of VPP Phy-to-Phy packet latency with IPv6 Overlay Tunnels measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶³.

3n-hsw-x520

78b-1t1c-base_and_scale

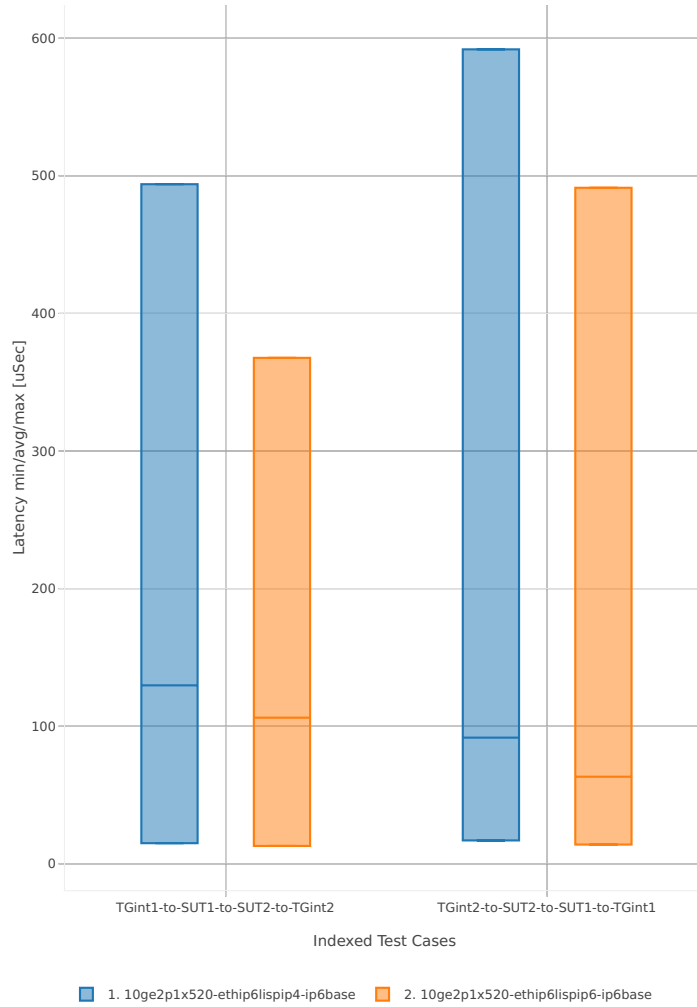
Latency: ip6tun-3n-hsw-x520-78b-1t1c-base_and_scale-ndr



⁶³ https://git.fd.io/csit/tree/tests/vpp/perf/ip6_tunnels?h=rls1807

78b-2t2c-base_and_scale

Latency: ip6tun-3n-hsw-x520-78b-2t2c-base_and_scale-ndr



2.5.7 KVM VMs vhost-user

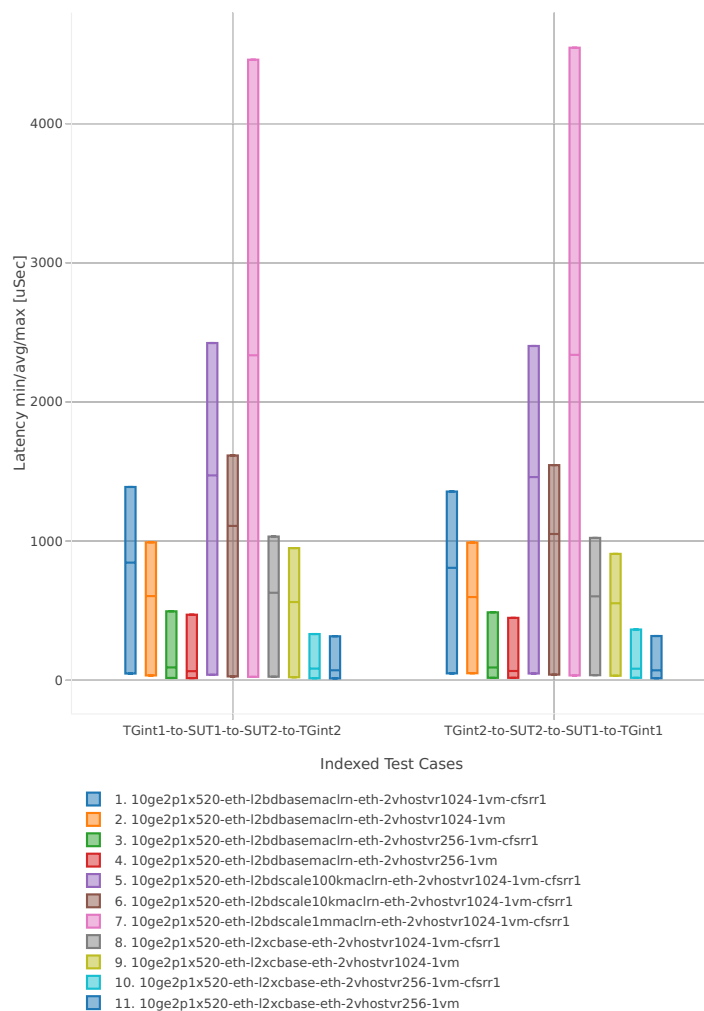
This section includes summary graphs of VPP Phy-to-VM(s)-to-Phy packet latency with with VM virtio and VPP vhost-user virtual interfaces measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶⁴.

3n-hsw-x520

64b-1t1c-base_and_scale-l2sw

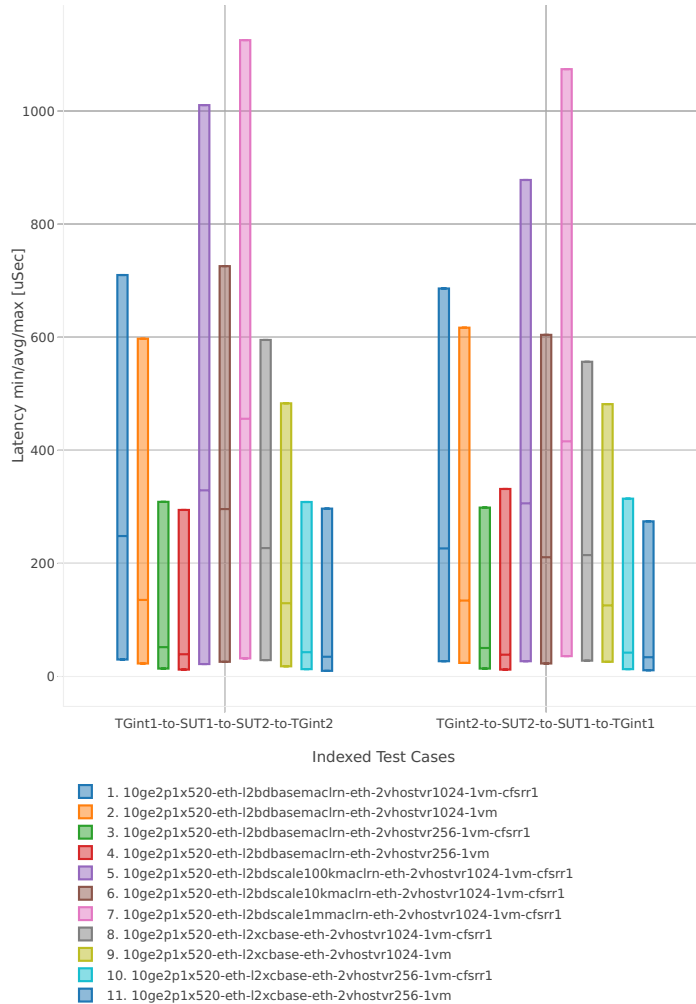
Latency: vhost-l2sw-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁶⁴ https://git.fd.io/csit/tree/tests/vpp/perf/vm_vhost?h=rls1807

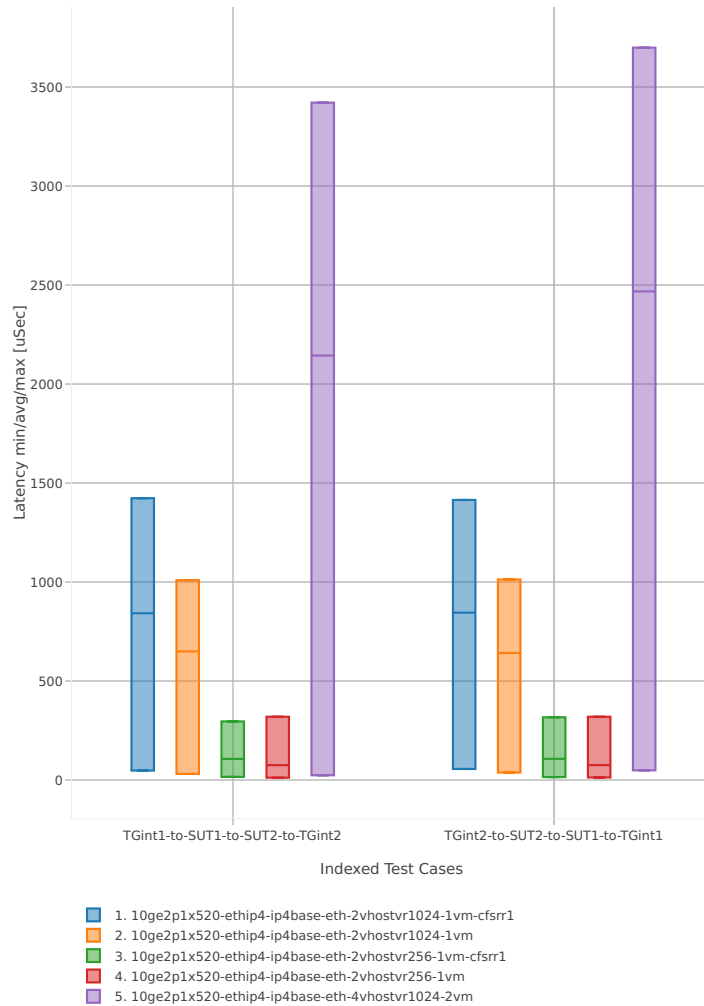
64b-2t2c-base_and_scale-l2sw

Latency: vhost-l2sw-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



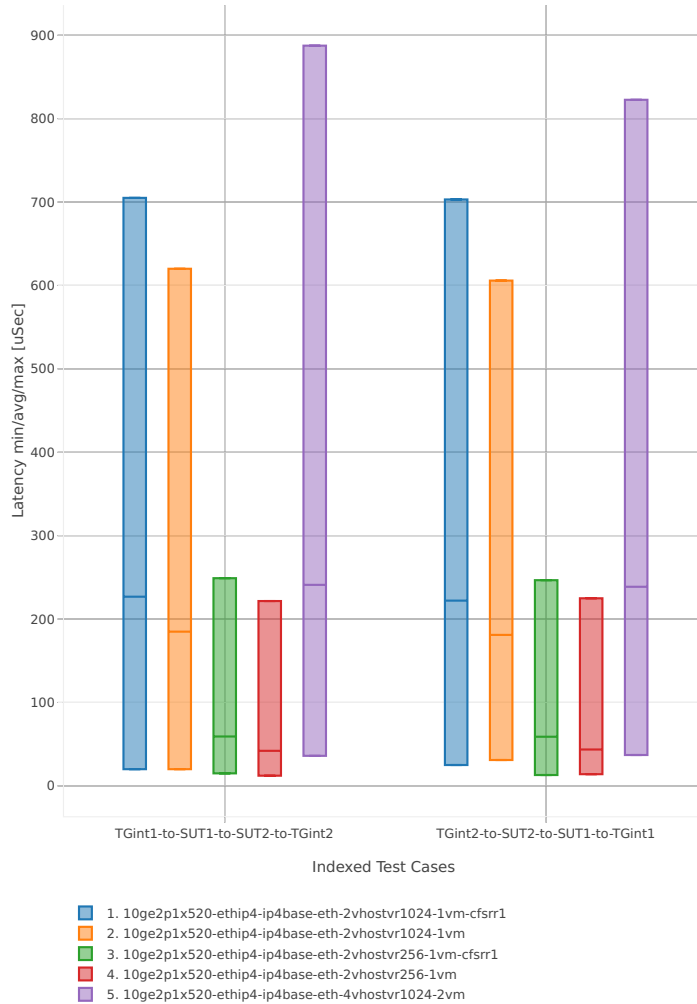
64b-1t1c-base_and_scale-ip4

Latency: vhost-ip4-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale-ip4

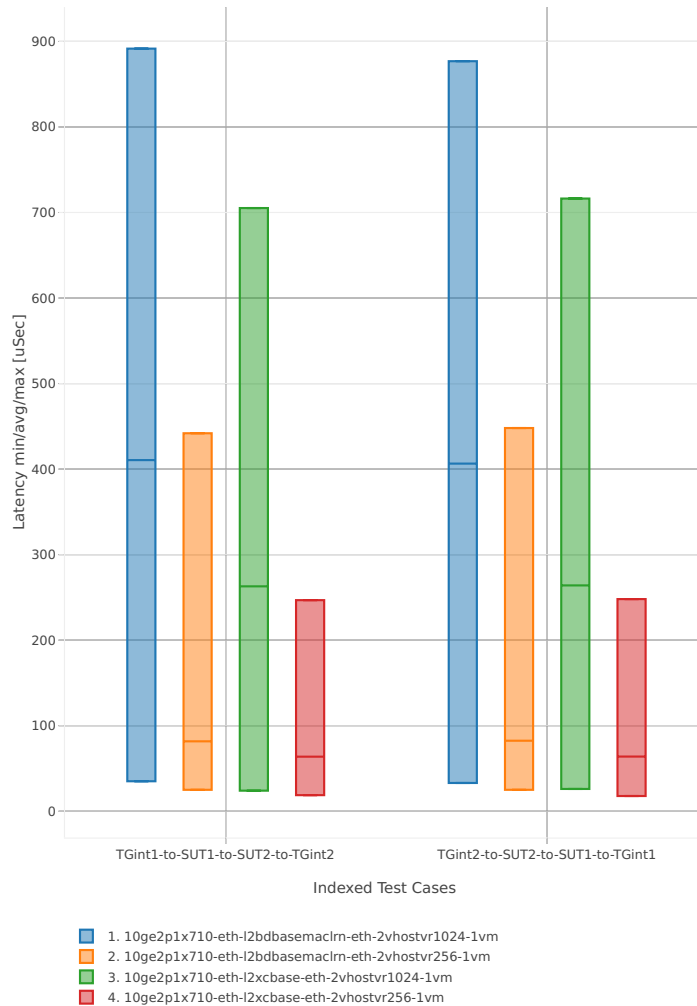
Latency: vhost-ip4-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



3n-hsw-x710

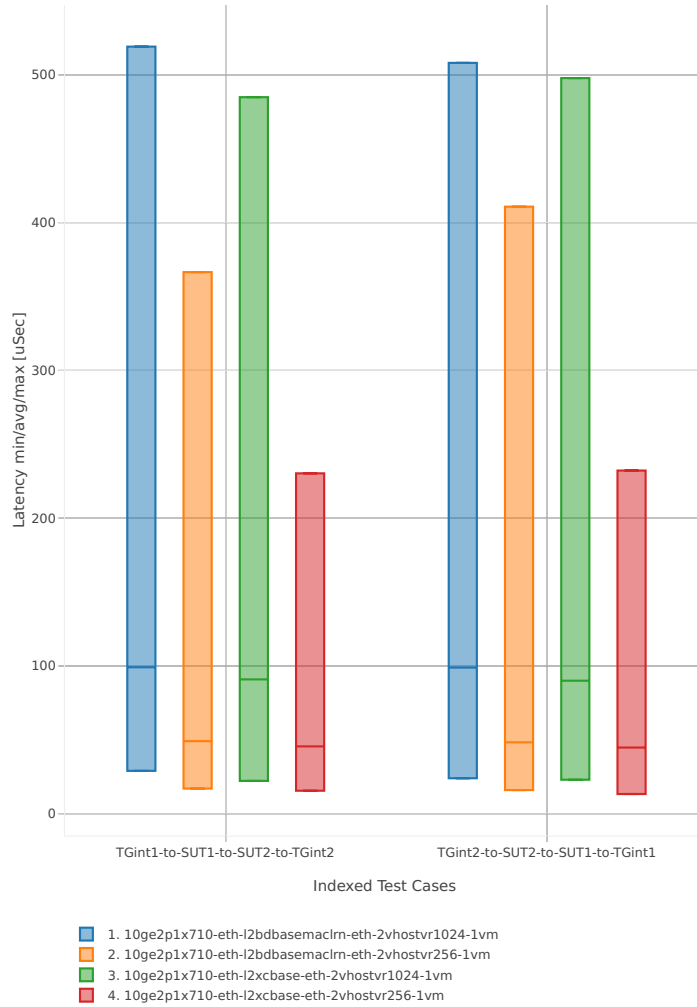
64b-1t1c-base_and_scale-l2sw

Latency: vhost-l2sw-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



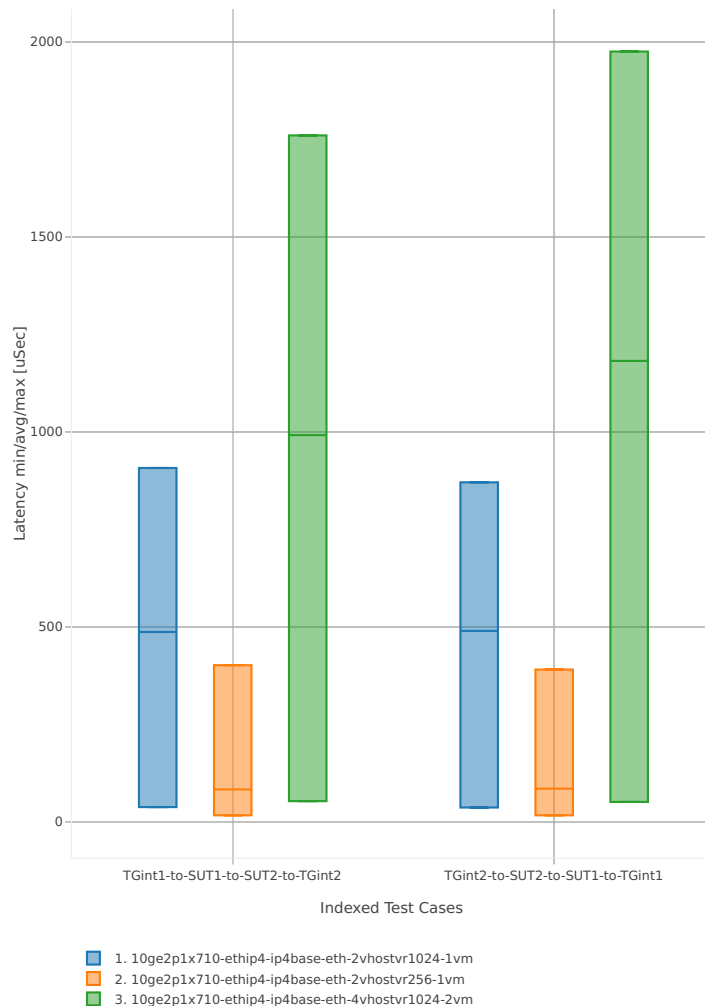
64b-2t2c-base_and_scale-l2sw

Latency: vhost-l2sw-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



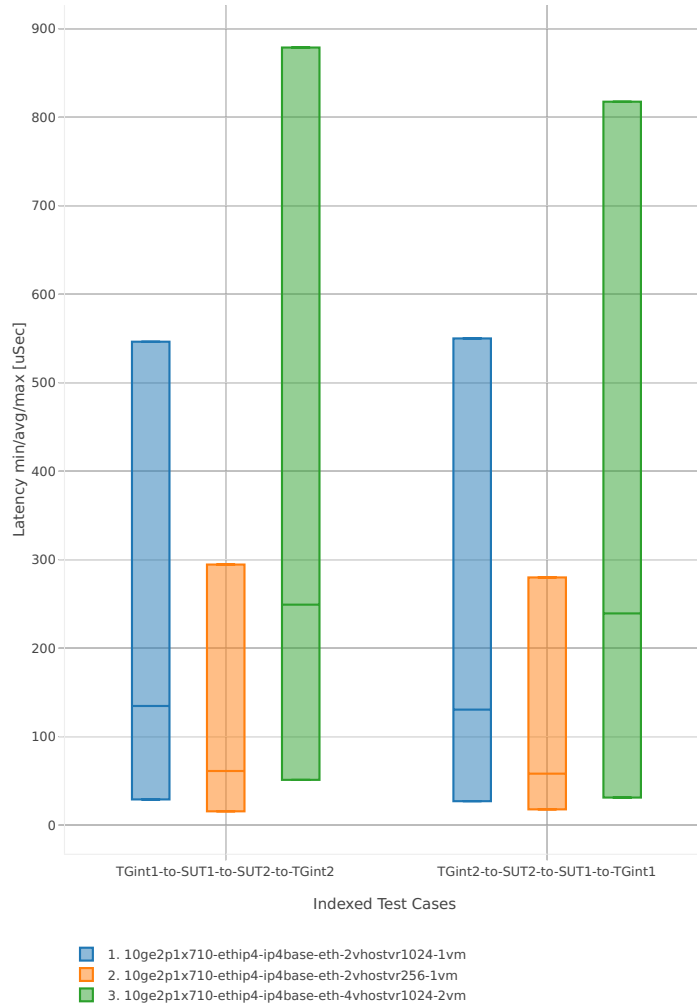
64b-1t1c-base_and_scale-ip4

Latency: vhost-ip4-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale-ip4

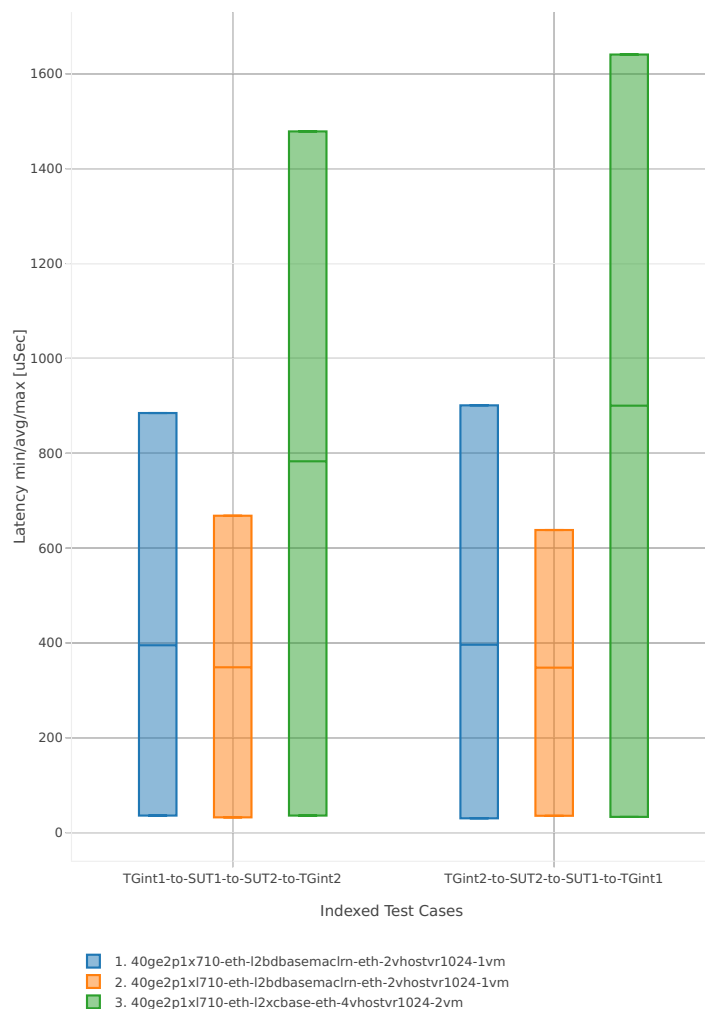
Latency: vhost-ip4-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



3n-hsw-xl710

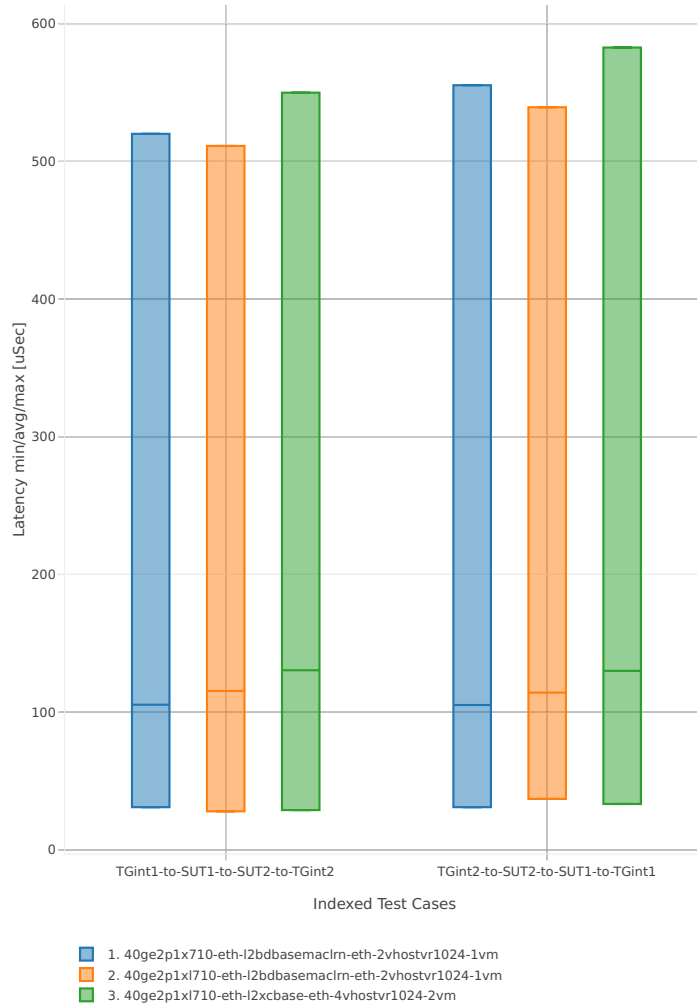
64b-1t1c-base_and_scale-l2sw

Latency: vhost-l2sw-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



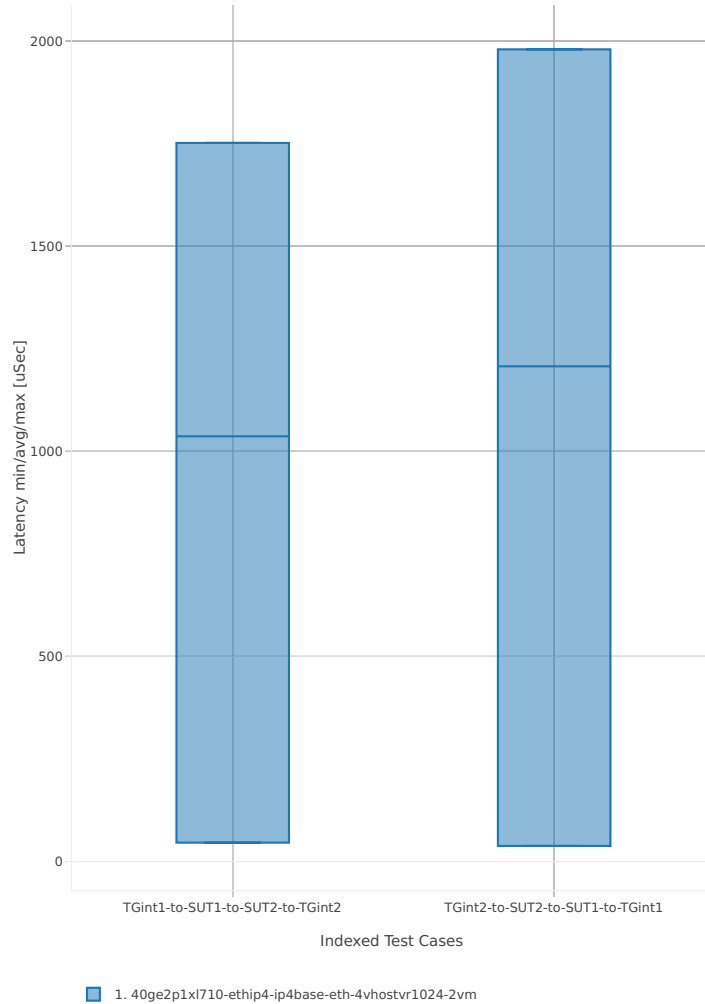
64b-2t2c-base_and_scale-l2sw

Latency: vhost-l2sw-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



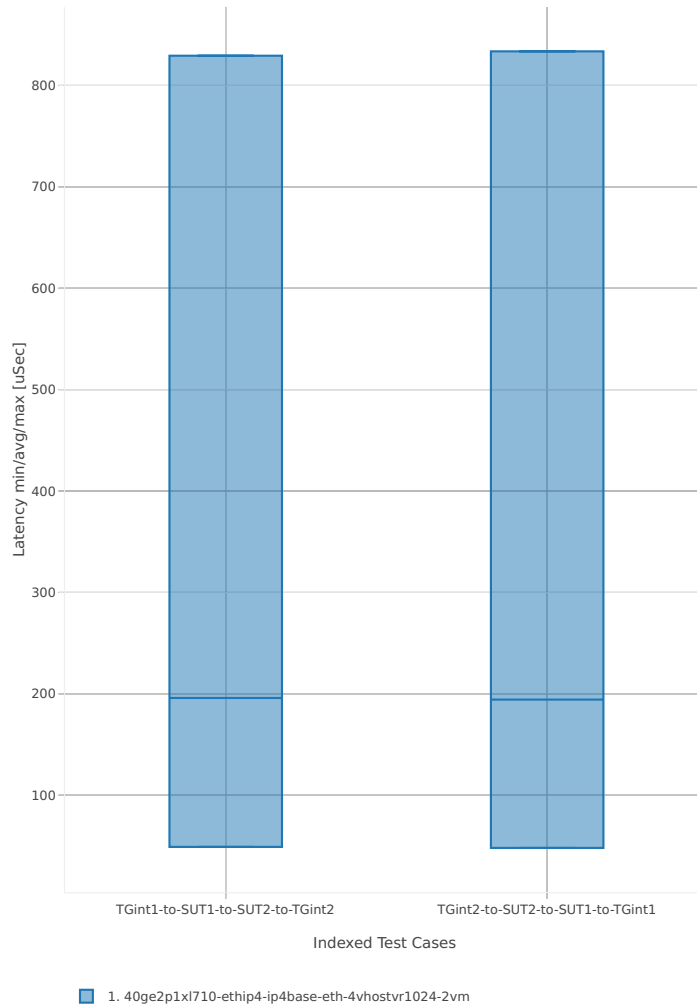
64b-1t1c-base_and_scale-ip4

Latency: vhost-ip4-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale-ip4

Latency: vhost-ip4-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



2.5.8 LXC/DRC Container Memif

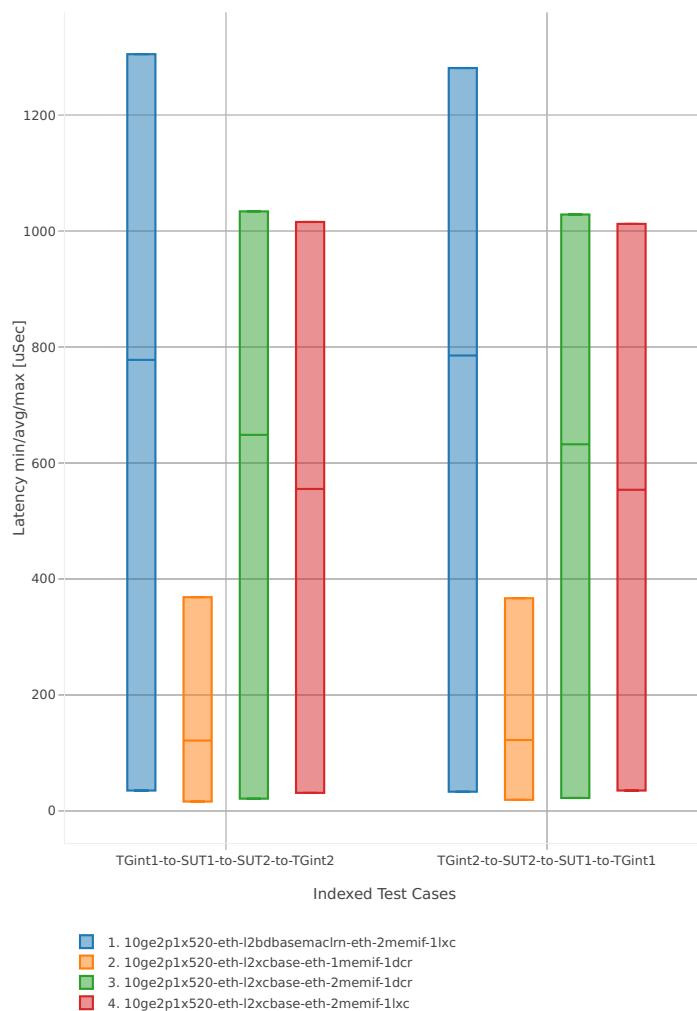
This section includes summary graphs of VPP Phy-to-Phy packet latency with Container memif Connections measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶⁵.

3n-hsw-x520

64b-1t1c-base_and_scale

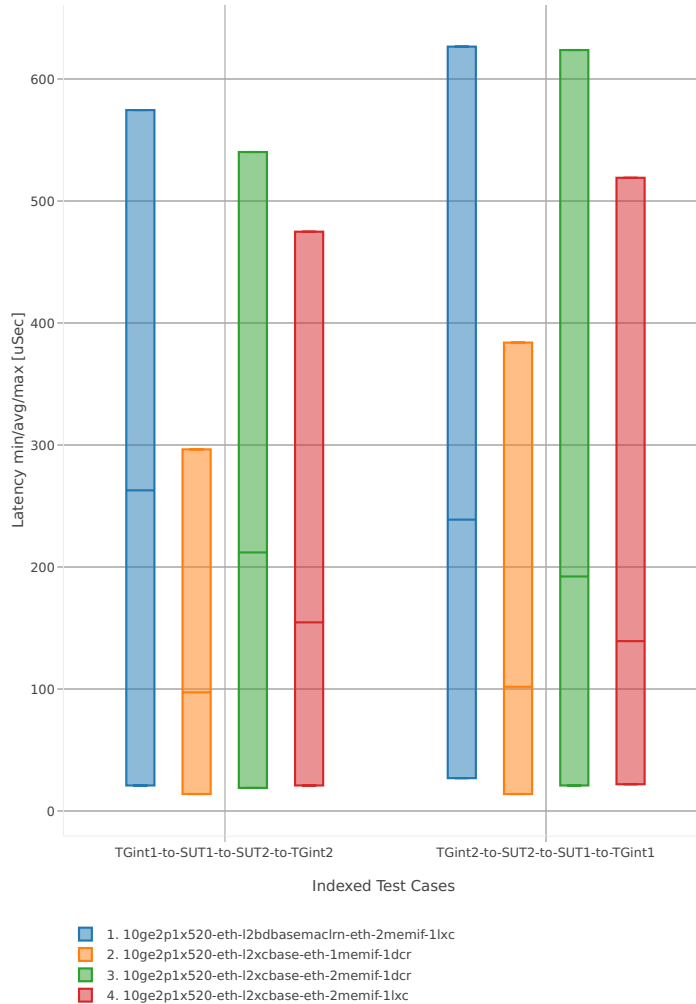
Latency: memif-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁶⁵ https://git.fd.io/csit/tree/tests/vpp/perf/container_memif?h=rls1807

64b-2t2c-base_and_scale

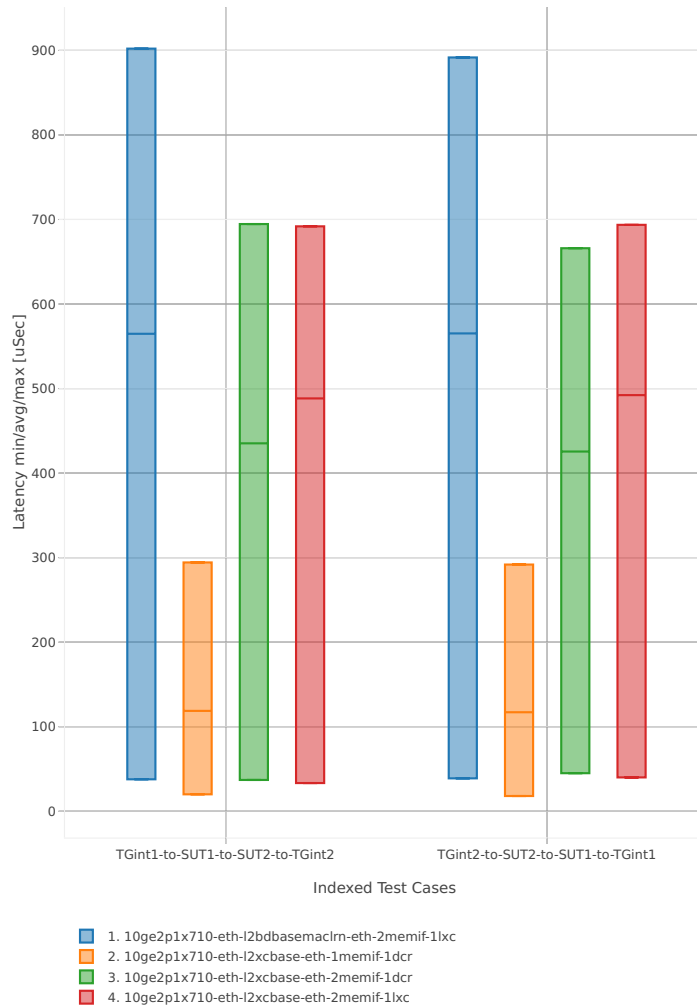
Latency: memif-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



3n-hsw-x710

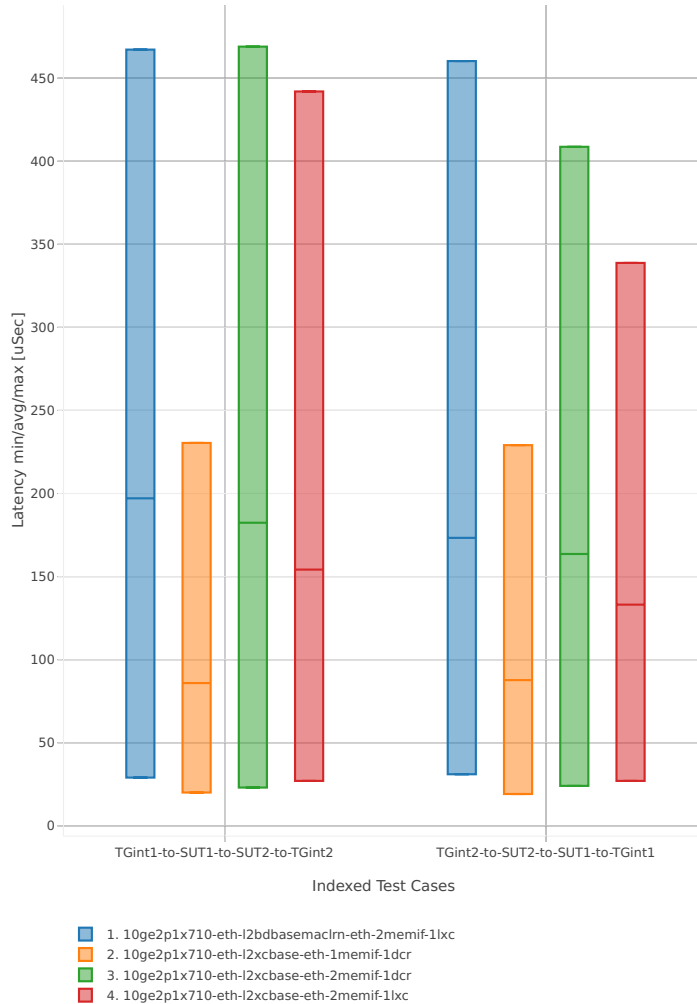
64b-1t1c-base_and_scale

Latency: memif-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

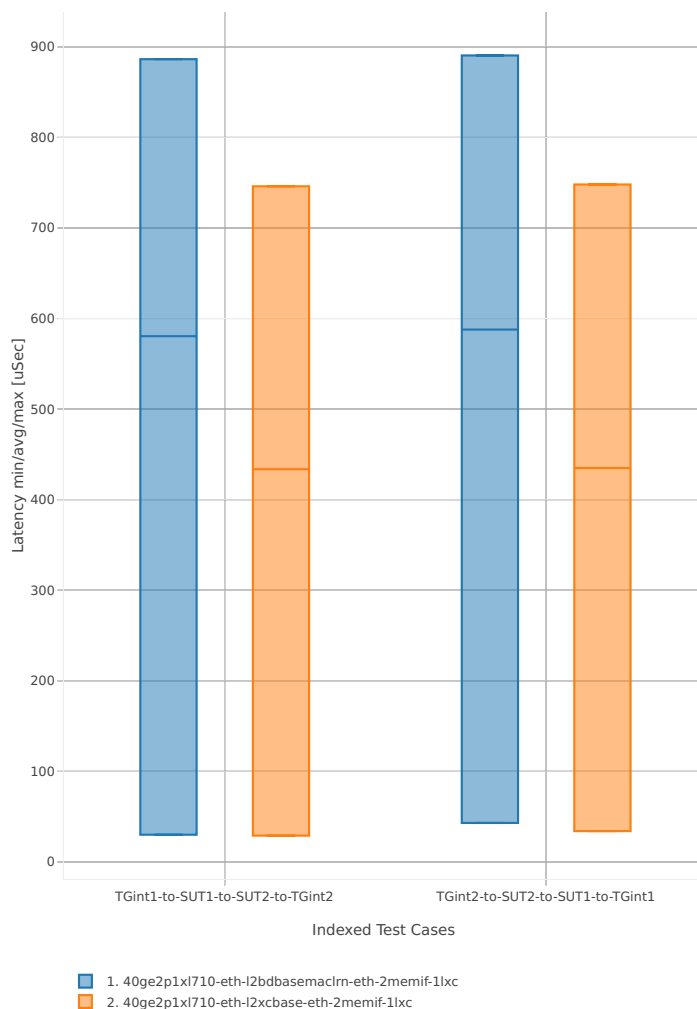
Latency: memif-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



3n-hsw-xl710

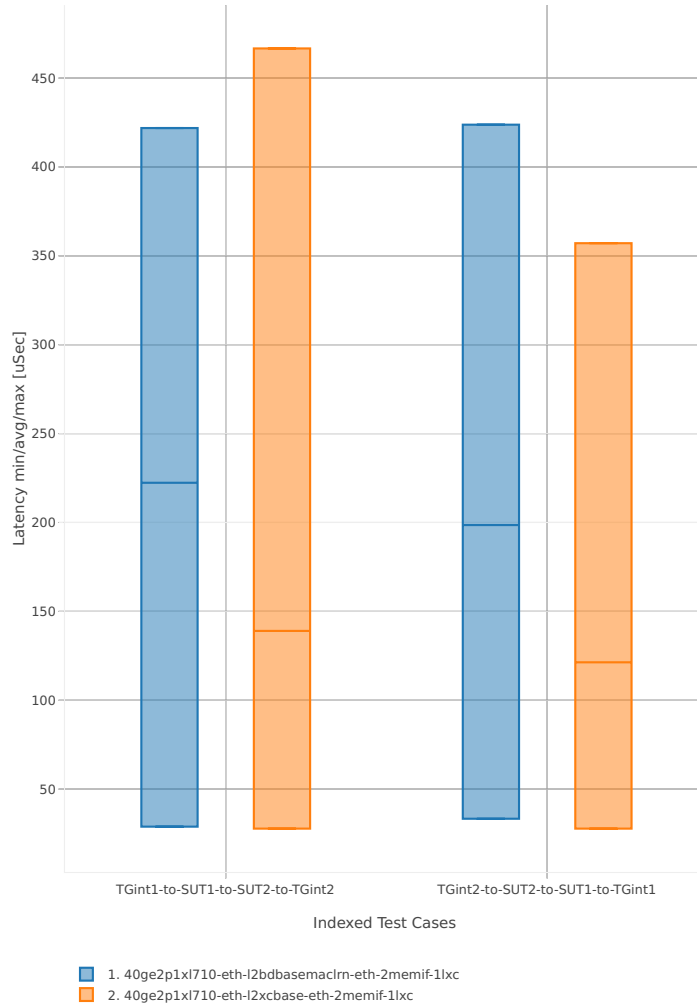
64b-1t1c-base_and_scale

Latency: memif-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

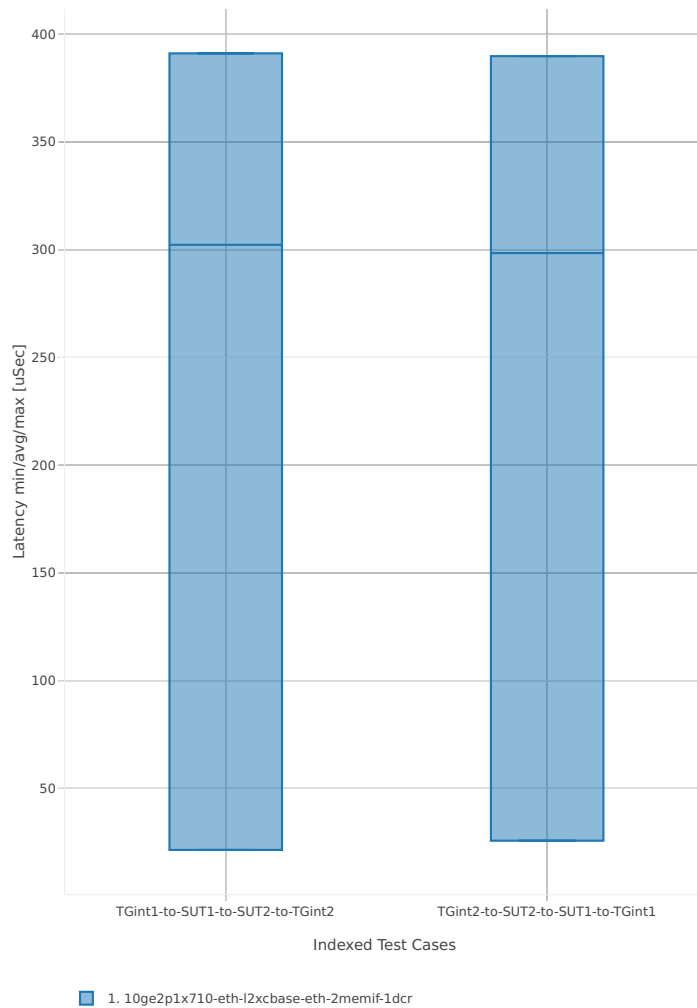
Latency: memif-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



3n-skx-x710

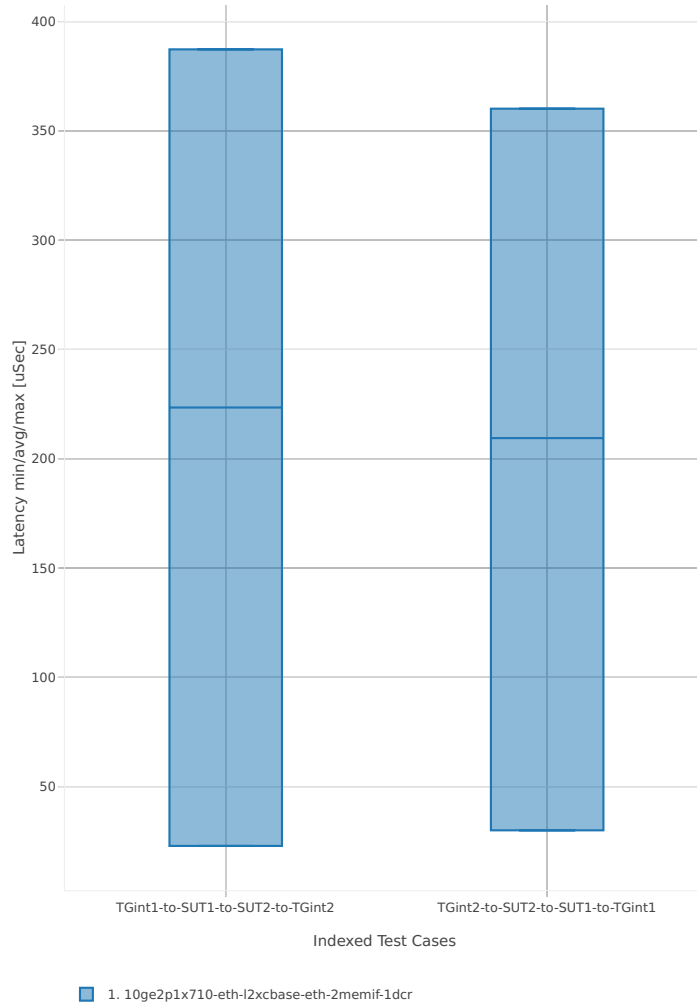
64b-2t1c-base_and_scale

Latency: memif-3n-skx-x710-64b-2t1c-base_and_scale-ndr



64b-4t2c-base_and_scale

Latency: memif-3n-skx-x710-64b-4t2c-base_and_scale-ndr



2.5.9 K8s Container Memif

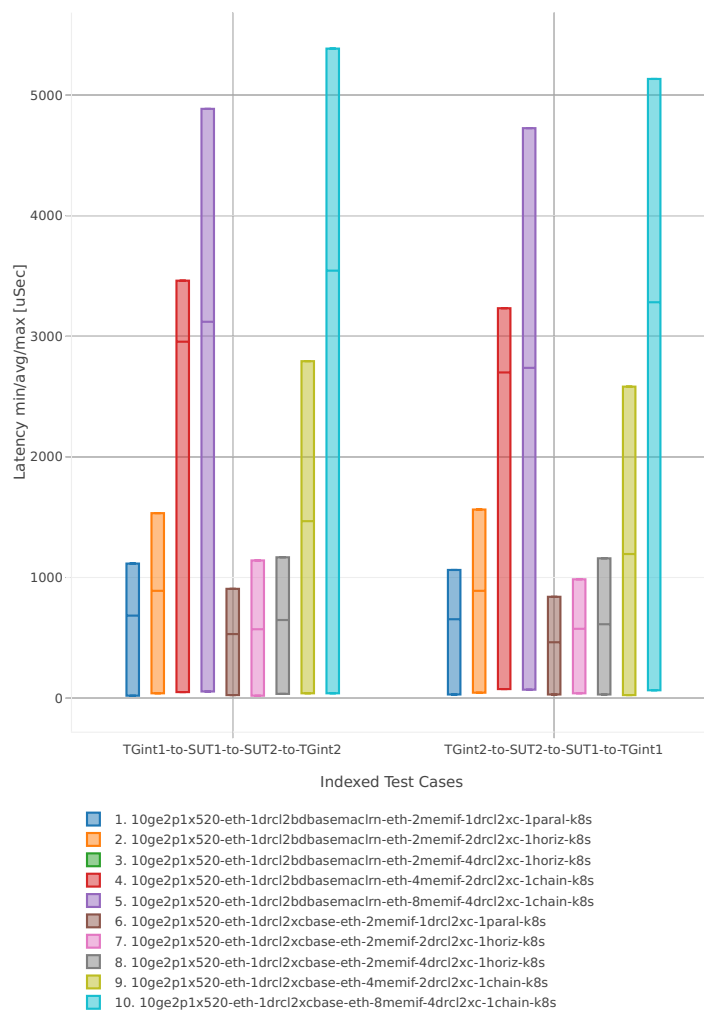
This section includes summary graphs of VPP Phy-to-Phy packet latency with CContainer Orchestrated Topologies measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶⁶.

3n-hsw-x520

64b-1t1c-base_and_scale

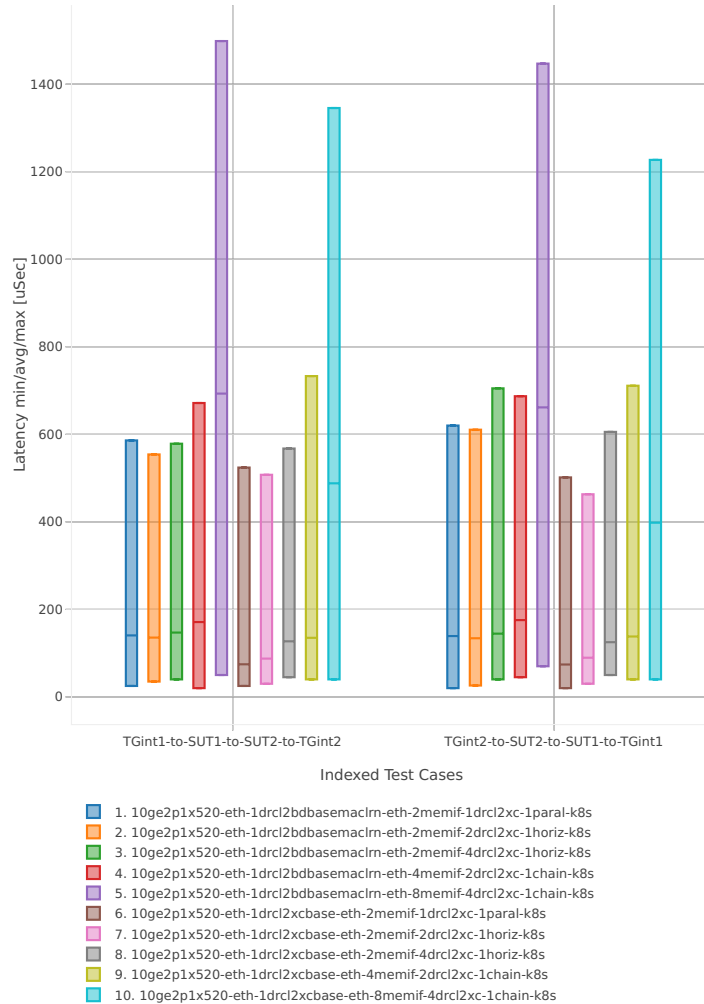
Latency: k8s-memif-3n-hsw-x520-64b-1t1c-base_and_scale-ndr



⁶⁶ https://git.fd.io/csit/tree/tests/kubernetes/perf/container_memif?h=rls1807

64b-2t2c-base_and_scale

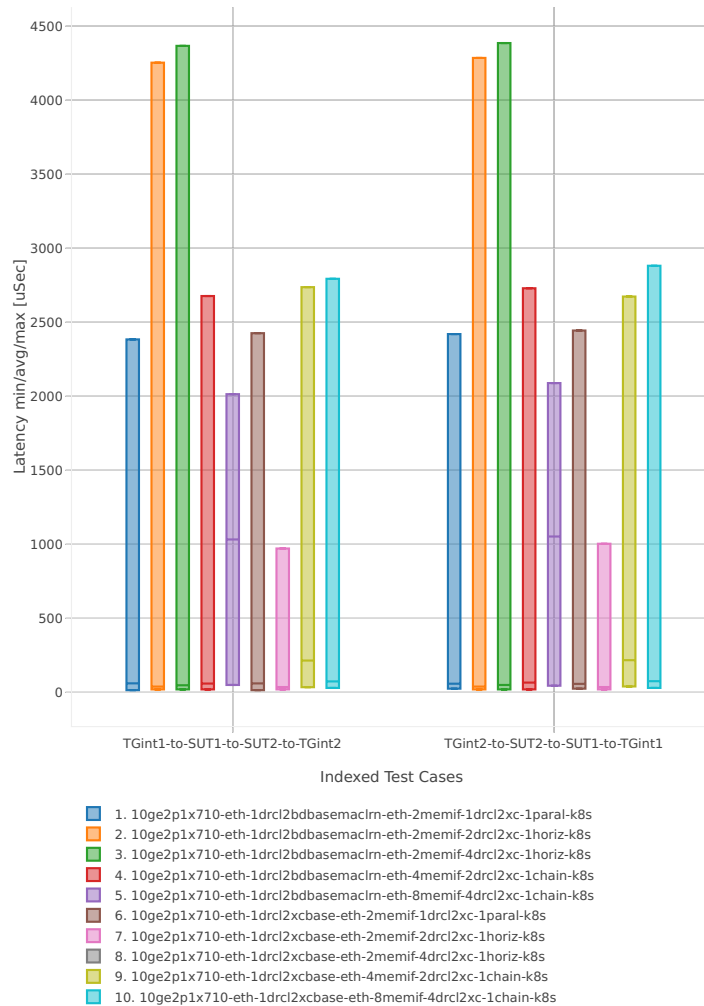
Latency: k8s-memif-3n-hsw-x520-64b-2t2c-base_and_scale-ndr



3n-hsw-x710

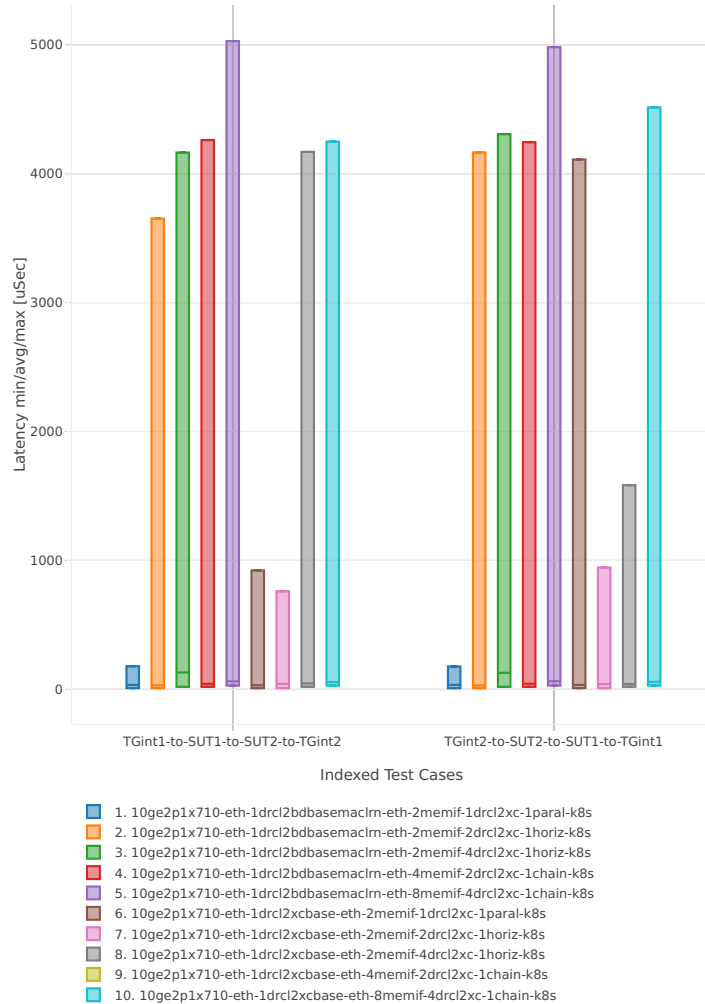
64b-1t1c-base_and_scale

Latency: k8s-memif-3n-hsw-x710-64b-1t1c-base_and_scale-ndr



64b-2t2c-base_and_scale

Latency: k8s-memif-3n-hsw-x710-64b-2t2c-base_and_scale-ndr



2.5.10 IPsec IPv4 Routing

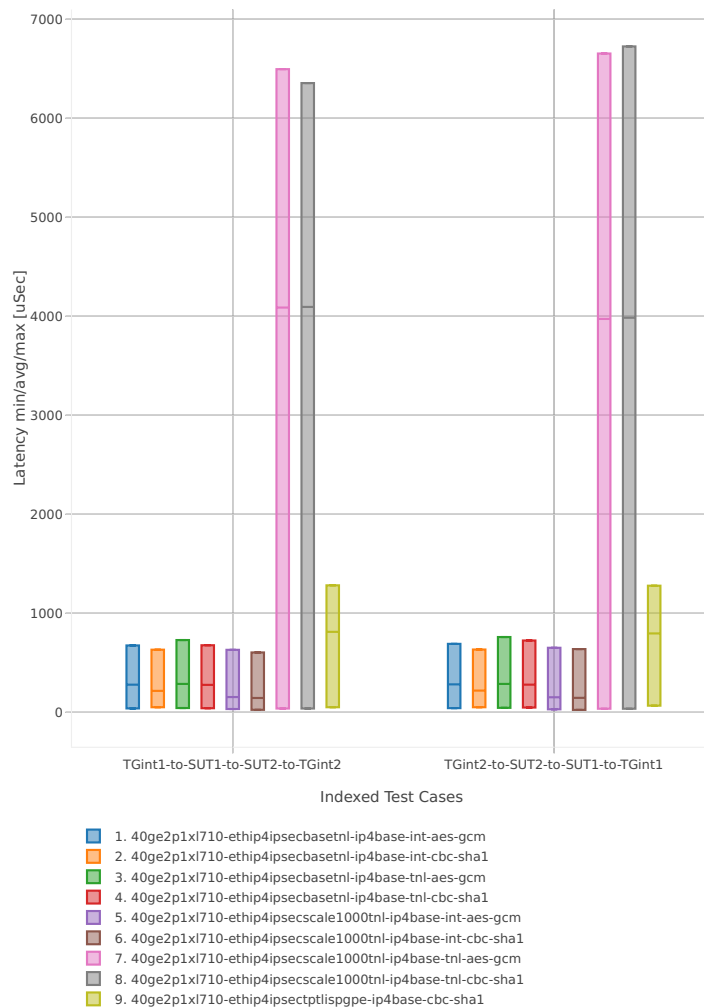
This section includes summary graphs of VPP Phy-to-Phy packet latency with IPsec encryption used in combination with IPv4 routed-forwarding, with latency measured at 100% of discovered NDR throughput rate. VPP IPsec encryption is accelerated using DPDK cryptodev library driving Intel Quick Assist (QAT) crypto PCIe hardware cards. Latency 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](#)⁶⁷.

3n-hsw-xl710

64b-1t1c-base_and_scale

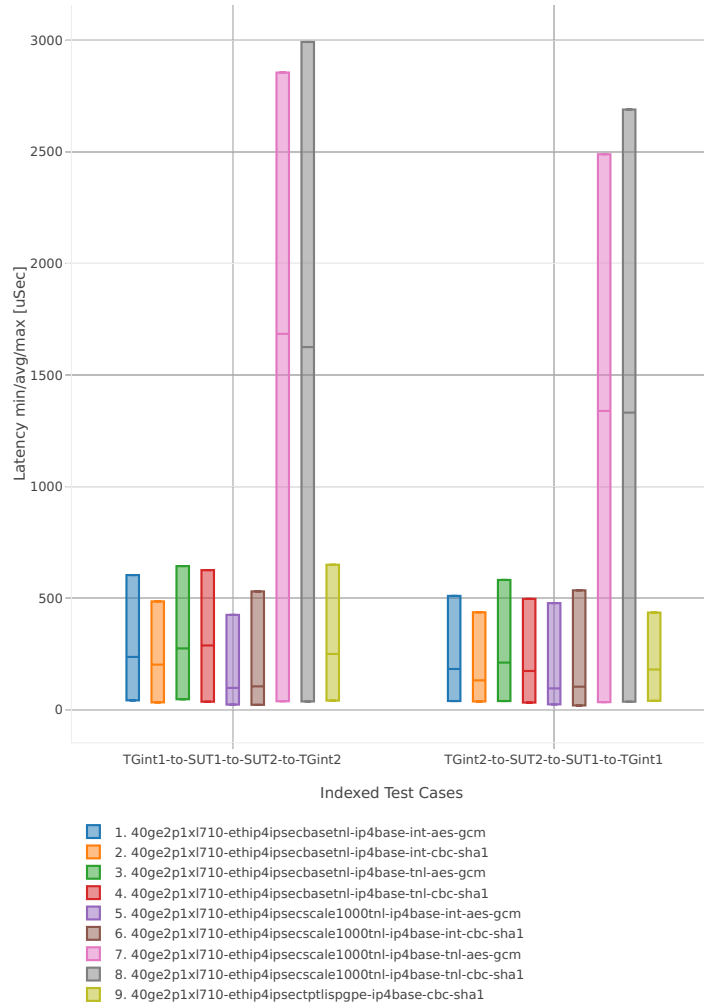
Latency: ipsec-3n-hsw-xl710-64b-1t1c-base_and_scale-ndr



⁶⁷ <https://git.fd.io/csit/tree/tests/vpp/perf/crypto?h=rls1807>

64b-2t2c-base_and_scale

Latency: ipsec-3n-hsw-xl710-64b-2t2c-base_and_scale-ndr



2.5.11 VTS

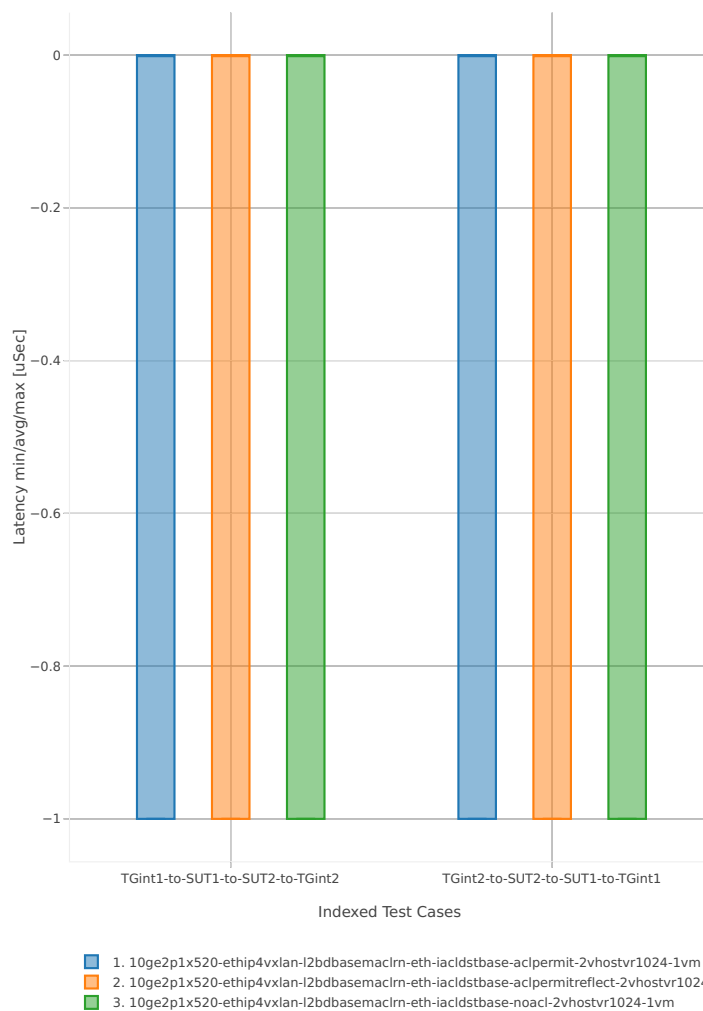
This section includes summary graphs of VPP Phy-to-VM(s)-to-Phy packet latency with with VM virtio and VPP vhost-user virtual interfaces measured at 100% of discovered NDR throughput rate. Latency is reported for VPP running in multiple configurations of VPP worker thread(s), a.k.a. VPP data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁶⁸.

3n-hsw-x520

114b-1t1c

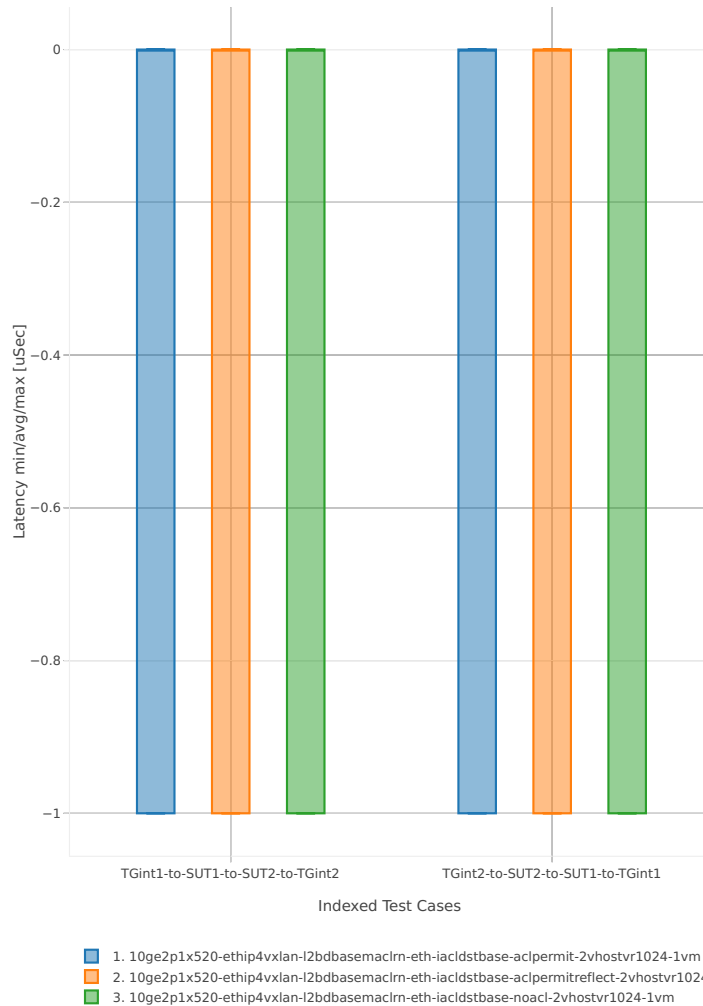
Latency: vts-3n-hsw-x520-114b-1t1c-ndr



⁶⁸ <https://git.fd.io/csit/tree/tests/vpp/perf/vts?h=rls1807>

114b-2t2c

Latency: vts-3n-hsw-x520-114b-2t2c-ndr



2.6 HTTP/TCP Performance

Performance graphs are generated by multiple executions of the same performance tests across physical testbeds hosted LF FD.io labs: 3n-hsw. Box-and-Whisker plots are used to display variations in measured throughput values, without making any assumptions of the underlying statistical distribution.

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.

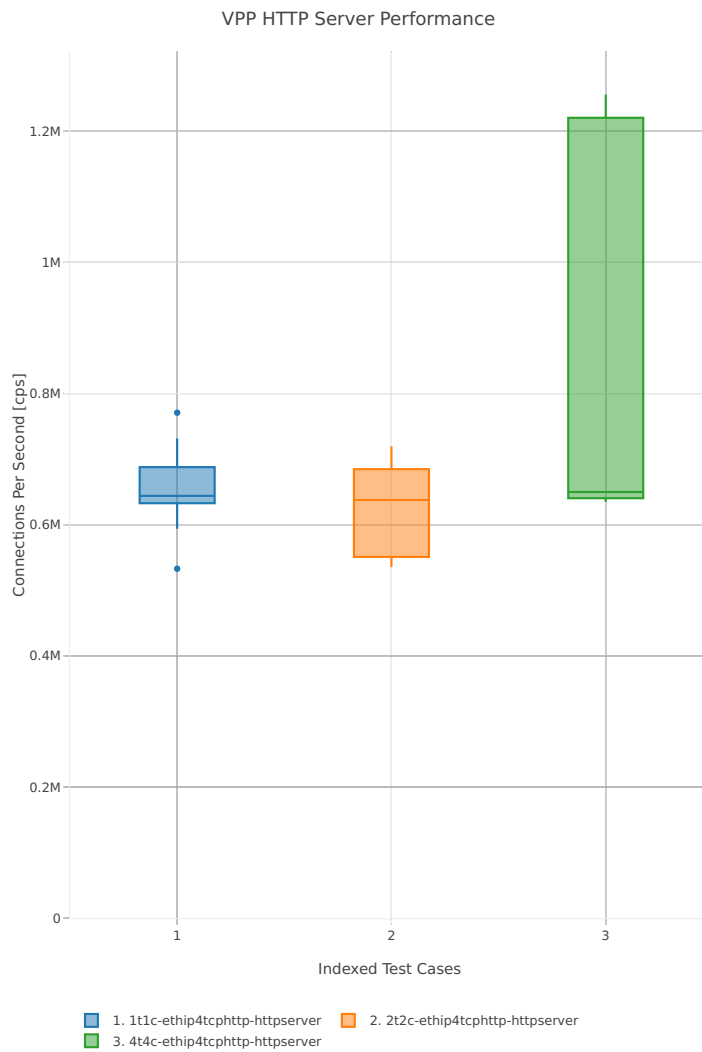
Note: Data sources for reported test results: i) [FD.io test executor vpp performance job 3n-hsw](#)⁶⁹, ii) archived FD.io jobs test result [output files](#).

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)⁷⁰.

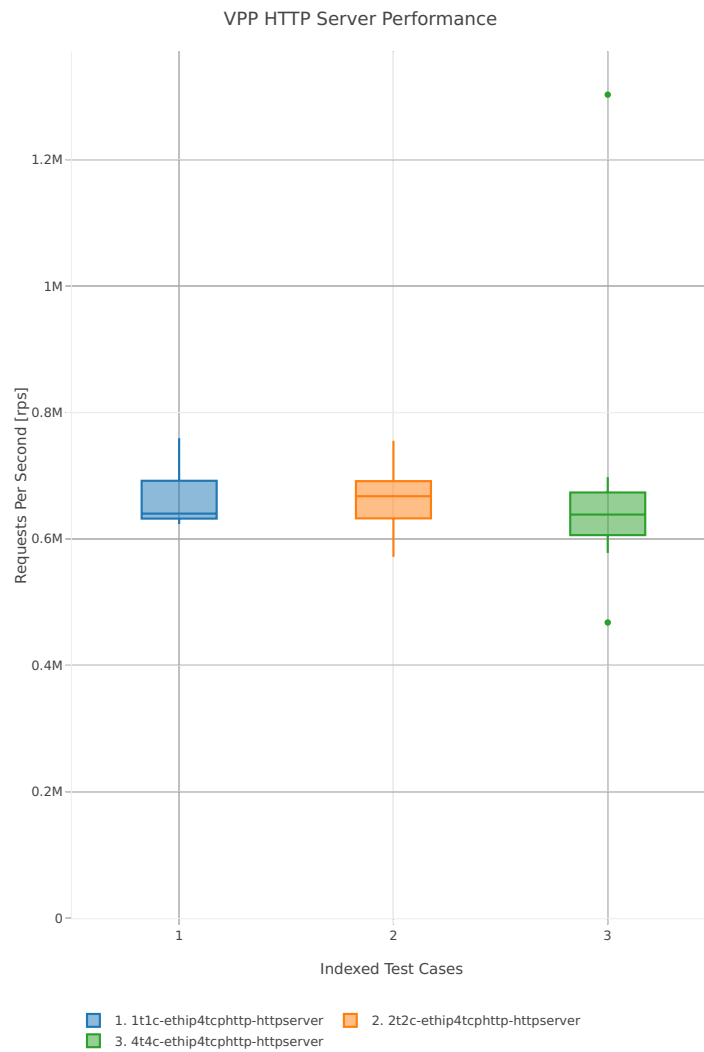
⁶⁹ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

⁷⁰ <https://git.fd.io/csit/tree/tests/vpp/perf/tcp?h=rls1807>

2.6.1 Connections per second



2.6.2 Requests per second



2.7 Comparisons

2.7.1 Current vs. Previous Release

Relative comparison of VPP packet throughput (NDR, PDR and MRR) between VPP-18.07 release and VPP-18.04 release (measured for CSIT-18.07 and CSIT-18.04 respectively) is calculated from results of tests running on 3-Node Intel Xeon Haswell testbeds (3n-hsw) 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 3n-hsw](#)⁷¹, with RF result files csit-vpp-perf-1807-*.zip [archived here](#).

NDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII 1t1c NDR comparison](#)
- [ASCII 2t2c NDR comparison](#)
- [CSV 1t1c NDR comparison](#)
- [CSV 2t2c NDR comparison](#)

PDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII 1t1c PDR comparison](#)
- [ASCII 2t2c PDR comparison](#)
- [CSV 1t1c PDR comparison](#)
- [CSV 2t2c PDR comparison](#)

MRR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII 1t1c MRR comparison](#)
- [ASCII 2t2c MRR comparison](#)
- [ASCII 4t4c MRR comparison](#)
- [CSV 1t1c MRR comparison](#)
- [CSV 2t2c MRR comparison](#)
- [CSV 4t4c MRR comparison](#)

⁷¹ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

2.7.2 3n-Skx vs. 3n-Hsw Testbeds

Relative comparison of VPP-18.07 release packet throughput (NDR, PDR and MRR) is calculated for the same tests executed on 3-Node Skylake (3n- skx) and 3-Node Haswell (3n-hsw) 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-hsw⁷²](#) and [FD.io test executor vpp performance job 3n-skx⁷³](#) with RF result files csit-vpp-perf-1807-*.zip [archived here](#).

NDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII NDR comparison](#)
- [CSV NDR comparison](#)

PDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII PDR comparison](#)
- [CSV PDR comparison](#)

MRR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII MRR comparison](#)
- [CSV MRR comparison](#)

2.8 Throughput Trending

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

1. [Performance Dashboard⁷⁴](#): per VPP test case throughput trend, trend compliance and summary of detected anomalies.
2. [Trending Methodology⁷⁵](#): throughput test metrics, trend calculations and anomaly classification (progression, regression).
3. [VPP Trendline Graphs⁷⁶](#): per VPP build MRR throughput measurements against the trendline with anomaly highlights and associated CSIT test jobs.

⁷² <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-hsw>

⁷³ <https://jenkins.fd.io/view/csit/job/csit-vpp-perf-verify-1807-3n-skx>

⁷⁴ <https://docs.fd.io/csit/master/trending/introduction/index.html>

⁷⁵ <https://docs.fd.io/csit/master/trending/methodology/index.html>

⁷⁶ <https://docs.fd.io/csit/master/trending/trending/index.html>

2.9 Test Environment

2.9.1 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, Arm, Intel Atom. More detailed description is provided in *Physical Testbeds* (page 4). Tested logical topologies are described in *Logical Topologies* (page 20).

2.9.2 Server Specifications

Complete technical specifications of compute servers used in CSIT physical testbeds are maintained on FD.io wiki pages: [CSIT/Testbeds: Xeon Hsw, VIRL](#)⁷⁷ and [CSIT Testbeds: Xeon Skx, Arm, Atom](#)⁷⁸.

2.9.3 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](#)⁷⁹.
2. Memory bandwidth – measure bandwidth with [Intel MLC tool](#)⁸⁰.
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 Intel Xeon Haswell and Intel Xeon Skylake testbeds.

2.9.4 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 [CSIT/Testbeds: Xeon Hsw, VIRL](#)⁸¹.

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

⁷⁷ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Hsw,_VIRL.#FD.io_CSIT_testbeds_-_Xeon_Haswell.2C_VIRL

⁷⁸ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.#Server_Specification

⁷⁹ https://git.fd.io/pma_tools/tree/jitter

⁸⁰ <https://software.intel.com/en-us/articles/intelr-memory-latency-checker>

⁸¹ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Hsw,_VIRL.#FD.io_CSIT_testbeds_-_Xeon_Haswell.2C_VIRL

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.4.0-72-generic root=UUID=efb7e8b3-3548-4440-98f6-6ebe102e9ec6 ro isolcpus=1-
↪17,19-35 nohz_full=1-17,19-35 rcu_nocbs=1-17,19-35 intel_pstate=disable console=tty0
↪console=ttyS0,115200n8
```

Linux uname

```
$ uname -a
Linux t3-sut2 4.4.0-72-generic #93-Ubuntu SMP Fri Mar 31 14:07:41 UTC 2017 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 program will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Timings are in CPU Core cycles
Inst_Min:   Minimum Execution time during the display update interval(default is ~1 second)
Inst_Max:   Maximum Execution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Execution time during the display update interval. This is the value of
↪interest
last_Exec:  The Execution time of last iteration just before the display update
Abs_Min:   Absolute Minimum Execution time since the program started or statistics were reset
Abs_Max:   Absolute Maximum Execution time since the program started or statistics were reset
tmp:       Cumulative value calculated 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								
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								

(continues on next page)

(continued from previous page)

	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
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
```

(continues on next page)

(continued from previous page)

```

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

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

```

(continues on next page)

(continued from previous page)

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    -
```

2.9.5 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 [CSIT Testbeds: Xeon Skx, Arm, Atom](#)⁸².

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

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=vmlinux-4.15.0-23-generic root=UUID=759ad671-ad46-441b-a75b-9f54e81837bb 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 use=off_
↪console=tty0 console=ttyS0,115200n8_
82 https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.#Server_Specification
```

(continues on next page)

Linux uname

```
$ uname -a
Linux s5-t22-sut1 4.15.0-23-generic #25-Ubuntu SMP Wed May 23 18:02:16 UTC 2018 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 program 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 calcaulted 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								
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								

(continues on next page)

(continued from previous page)

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
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
```

(continues on next page)

(continued from previous page)

```
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
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  -

```

2.9.6 SUT Settings - Linux

System provisioning is done by combination of PXE boot unattended install and [Ansible](#)⁸³ described in [CSIT Testbed Setup](#)⁸⁴.

Below a subset of the running configuration:

1. Xeon Haswell - Ubuntu 16.04.1 LTS

```

$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:       16.04
Codename:      xenial

```

2. Xeon Skylake - Ubuntu 18.04 LTS

```

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

```

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>

⁸³ <https://www.ansible.com>

⁸⁴ <https://git.fd.io/csit/tree/resources/tools/testbed-setup/README.md?h=rls1807>

- **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.

Applied Boot Cmdline

1. Xeon Haswell - Ubuntu 16.04.1 LTS

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.4.0-72-generic root=UUID=35ea11e4-e44f-4f67-8cbe-12f09c49ed90 ro isolcpus=1-
↪17,19-35 nohz_full=1-17,19-35 rcu_nocbs=1-17,19-35 intel_pstate=disable console=tty0_
↪console=ttyS0,115200n8
```

2. Xeon Skylake - Ubuntu 18.04 LTS

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.15.0-23-generic root=UUID=3fa246fd-1b80-4361-bb90-f339a6bbbed51 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 CFS tunings

Linux CFS scheduler tunings are applied to all QEMU vCPU worker threads (the ones handling testpmd PMD threads) and VPP data plane worker threads. List of VPP data plane threads can be obtained by running:

```
$ for psid in $(pgrep vpp)
$ do
$   for tid in $(ps -Lo tid --pid $psid | grep -v TID)
$   do
$     echo $tid
$   done
$ done
```

Or:

```
$ cat /proc/`pidof vpp`/task/*/stat | awk '{print $1" "$2" "$39}'
```

CFS round-robin scheduling with highest priority is applied using:

```
$ for psid in $(pgrep vpp)
$ do
$   for tid in $(ps -Lo tid --pid $psid | grep -v TID)
$     do
$       chrt -r -p 1 $tid
$     done
$ done
```

More information about Linux CFS can be found in [Sched manual pages](#)⁸⁵.

Host IRQ Affinity

IRQs are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ for l in `ls /proc/irq`; do echo 1 | sudo tee /proc/irq/$l/smp_affinity; done
```

Host RCU Affinity

RCUs are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ for i in `pgrep rcu[^c]`; do sudo taskset -pc 0 $i ; done
```

Host Writeback Affinity

Writebacks are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ echo 1 | sudo tee /sys/bus/workqueue/devices/writeback/cpumask
```

2.9.7 DUT Settings - VPP

VPP Version

VPP-18.07 release

VPP Compile Parameters

FD.io VPP compile job⁸⁶

VPP Install Parameters

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

⁸⁵ <http://man7.org/linux/man-pages/man7/sched.7.html>

⁸⁶ <https://jenkins.fd.io/view/vpp/job/vpp-merge-1807-ubuntu1604/>

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
}
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 $$NUM_RX_QUEUES
  }
  socket-mem 1024,1024
  no-tx-checksum-offload
  dev $$DEV_1
  dev $$DEV_2
}
```

Description of VPP startup settings used in CSIT is provided in *Performance Test Methodology* (page 9).

2.9.8 TG Settings - TRex

TG Version

TRex v2.35

DPDK version

DPDK v17.11

TG Build Script used

TRex intallation⁸⁷

TG Startup Configuration

```
$ cat /etc/trex_cfg.yaml
- port_limit      : 2
  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 --iom 0 > /tmp/trex.log 2>
↵&1 &' > /dev/null
```

TG API Driver

TRex driver⁸⁸

⁸⁷ https://git.fd.io/csit/tree/resources/tools/trex/trex_installer.sh?h=rls1807

⁸⁸ https://git.fd.io/csit/tree/resources/tools/trex/trex_stateless_profile.py?h=rls1807

2.10 Documentation

2.10.1 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]* (page 582), *[lxcnamespace]* (page 582) and *[stgraber]* (page 582).

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]* (page 582) and *[stgraber]* (page 582).

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]* (page 582).
- Control groups - cgroups, resource bean counting, resource quotas, access restrictions, *[cgroup1]* (page 582), *[cgroup2]* (page 582).
- AppArmor - apparmor profiles aim to prevent any of the known ways of escaping a container or cause harm to the host, *[apparmor]* (page 582).
- 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]* (page 582).
- Seccomp - secure computing mode, enables filtering of system calls, *[seccomp]* (page 582).

More information in *[lxcsecurity]* (page 582) and *[lxcsecfeatures]* (page 582).

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/uisv_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](#)⁸⁹.
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.
- Combination of Kubernetes (container orchestration), Docker (container images) and Ligato (container networking).

LXC

LXC is the well-known and heavily tested low-level Linux container runtime [[lxcsource](#)] (page 582), 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 in 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 in `vpp startup.conf`.

⁸⁹ <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] (page 582). Currently it using *runc* a CLI tool for spawning and running containers according to the [OCI specification](#)⁹⁰

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.

Kubernetes

Kubernetes [k8sdoc] (page 582), or K8s, is a production-grade container orchestration platform for automating the deployment, scaling and operating application containers. Kubernetes groups containers that make up an application into logical units, pods, for easy management and discovery. K8s pod definitions including compute resource allocation is provided in .yaml files.

CSIT uses K8s and its infrastructure components like etcd to control all phases of container based virtualized network topologies.

Ligato

Ligato [ligato] (page 582) is an open-source project developing a set of cloud-native tools for orchestrating container networking. Ligato integrates with FD.io VPP using goVPP [govpp] (page 582) and vpp-agent [vppagent] (page 582).

Known Issues

- Currently using a separate LF Jenkins job for building csit-centric prod_vpp_agent docker images vs. dockerhub/ligato ones.

Implementation

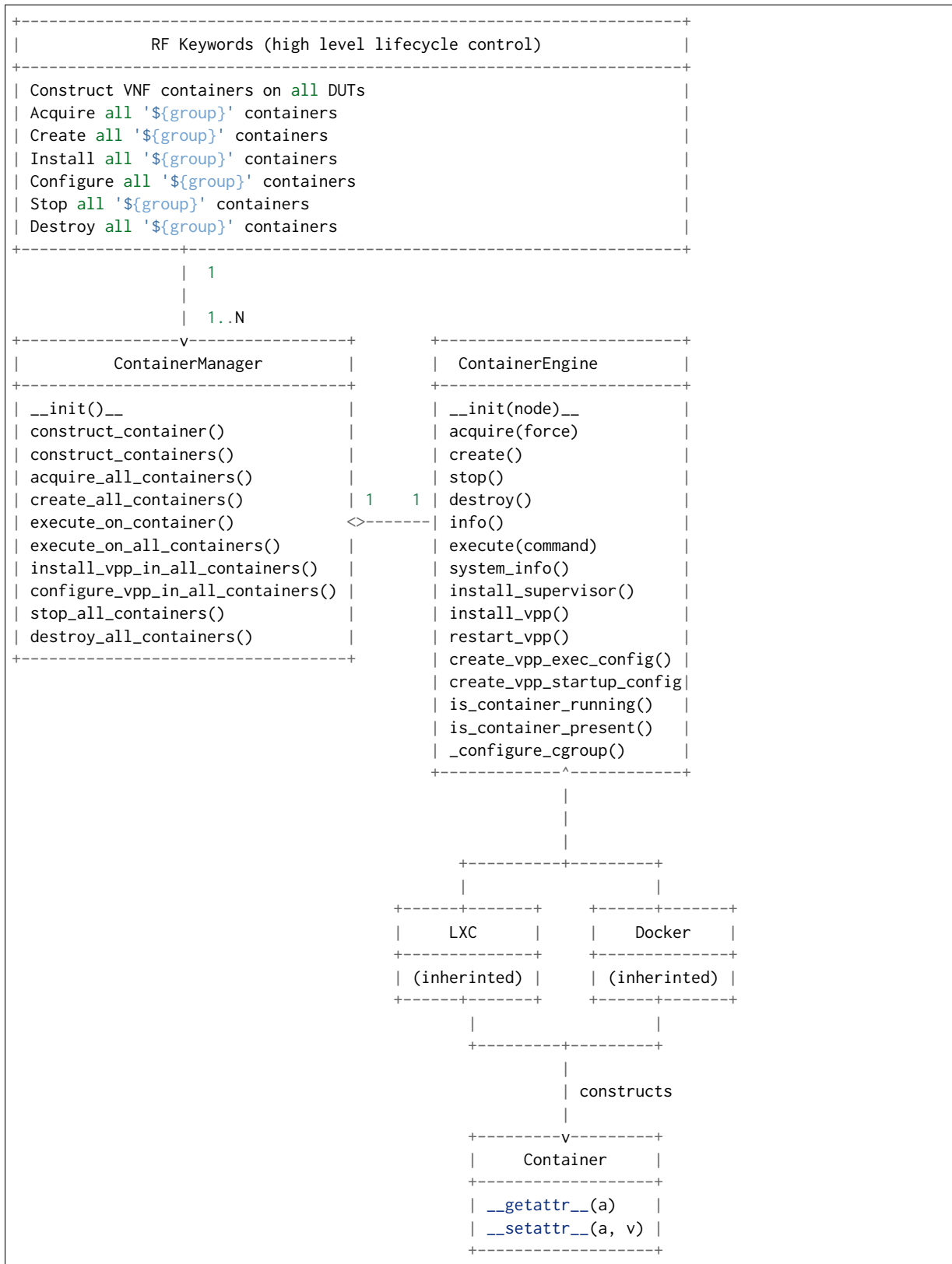
CSIT container orchestration is implemented in CSIT Level-1 keyword Python libraries following the Builder design pattern. Builder design pattern separates the construction of a complex object from its representation, so that the same construction process can create different representations e.g. LXC,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

⁹⁰ <https://www.opencontainers.org/>

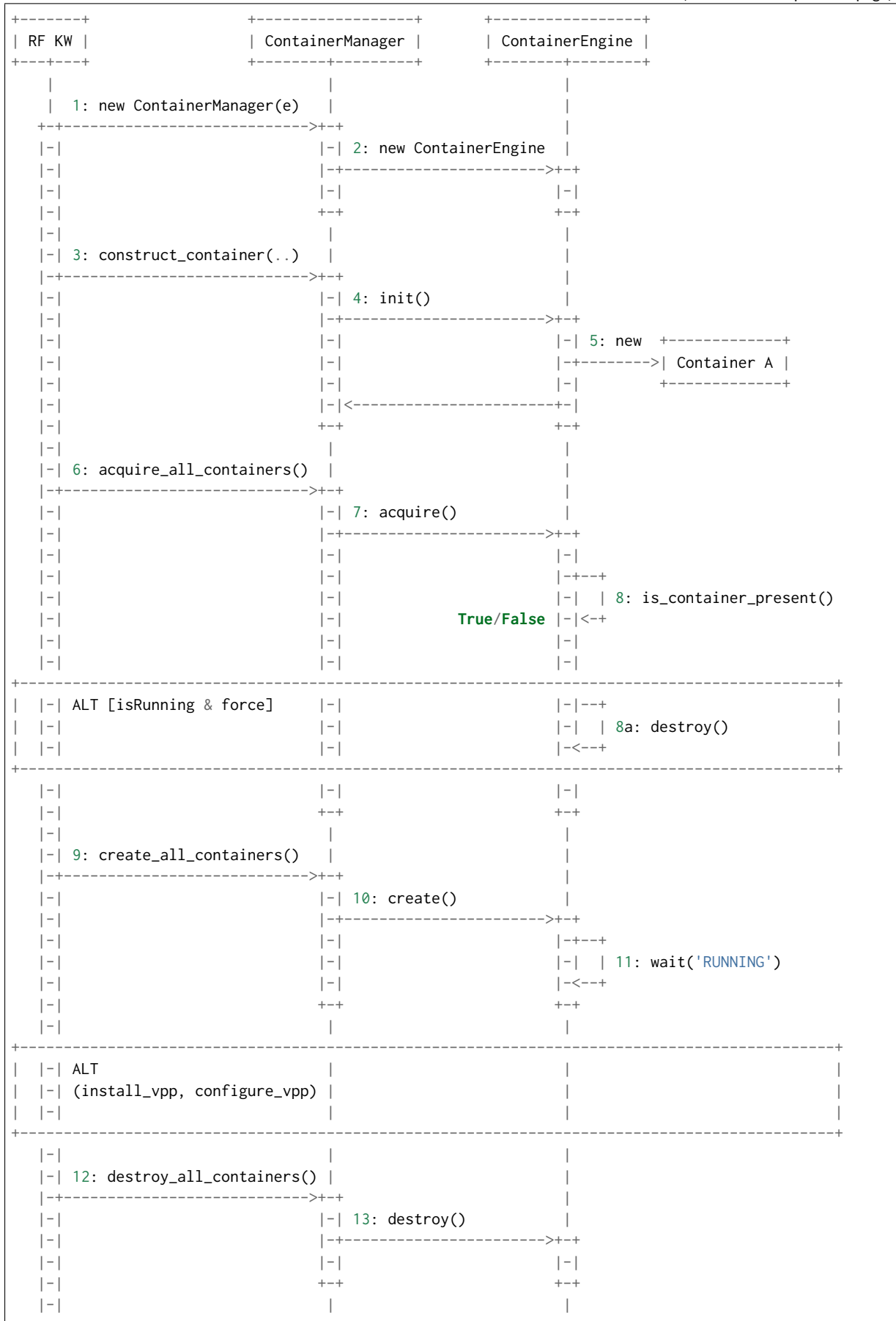


Sequential diagram that illustrates the creation of a single container.

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

(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)

+++		
+	+	+

Container Data Structure

Container is represented in Python L1 library as a separate Class with instance variables and no methods except overridden `__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}
| | | ${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*] (page 583), 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

Kubernetes 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 `kubect1` installed on each of DUTs. One time initialization script, `resources/libraries/bash/k8s_setup.sh` does reset/init `kubect1`, applies Calico v2.6.3 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. The templates can be found in `resources/templates/kubernetes` and can be leveraged in the future for other separate tasks.

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 e.g. `SFC_controller`, `VNF`, `VSWITCH`.

Making own python wrapper library of kubect1 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.

Ligato

Ligato integration does require to compile the vpp-agent tool and build the bundled Docker image. Compilation of vpp-agent depends on specific VPP. In ligato/vpp-agent repository there are well prepared scripts for building the Docker image. Building docker image is possible via series of commands:

```
git clone https://github.com/ligato/vpp-agent
cd vpp_agent/docker/dev_vpp_agent
sudo docker build -t dev_vpp_agent --build-arg AGENT_COMMIT=<agent commit id>\
--build-arg VPP_COMMIT=<vpp commit id> --no-cache .
sudo ./shrink.sh
cd ../prod_vpp_agent
sudo ./build.sh
sudo ./shrink.sh
```

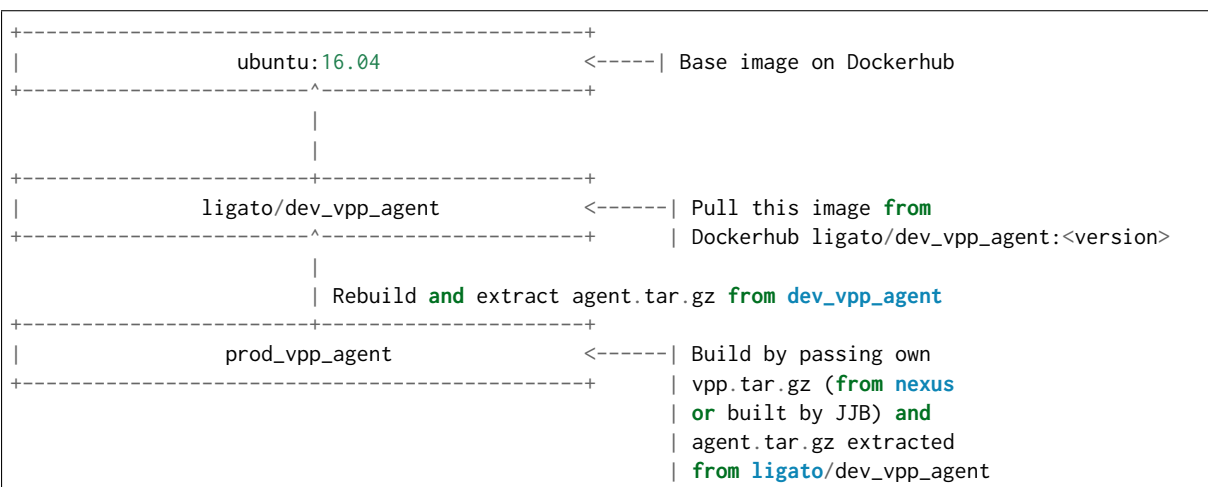
CSIT requires Docker image to include the desired VPP version (per patch testing, nightly testing, on demand testing).

The entire build process of building dev_vpp_agent image heavily depends on internet connectivity and also takes a significant amount of time (~1-1.5h based on internet bandwidth and allocated resources). The optimal solution would be to build the image on jenkins slave, transfer the Docker image to DUTs and execute separate suite of tests.

To adress the amount of time required to build dev_vpp_agent image, we can pull existing specific version of `dev_vpp_agent` and extract the `vpp-agent` from it.

We created separate sets of Jenkins jobs, that will be executing following:

1. Clone latest CSIT and Ligato repositories.
2. Pull specific version of dev_vpp_agent image from Dockerhub.
3. Extract VPP API (from .deb package) and copy into dev_vpp_agent image
4. Rebuild vpp-agent and extract outside image.
5. Build prod_vpp_image Docker image from dev_vpp_agent image.
6. Transfer prod_vpp_agent image to DUTs.
7. Execute subset of performance tests designed for Ligato testing.



Approximate size of vnf-agent docker images:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
dev-vpp-agent	latest	78c53bd57e2	6 weeks ago	9.79GB
prod_vpp_agent	latest	f68af5afe601	5 weeks ago	443MB

In CSIT we need to create separate performance suite under `tests/kubernetes/perf` which contains modified Suite setup in comparison to standard perf tests. This is due to reason that VPP will act as vswitch in Docker image and not as standalone installed service.

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-18.07:

- LXC topologies:
 - eth-l2xcbase-eth-2memif-1lxc.
 - eth-l2bdbasemaclrn-eth-2memif-1lxc.
- Docker topologies:
 - eth-l2xcbase-eth-2memif-1docker.
 - eth-l2xcbase-eth-1memif-1docker
- Kubernetes/Ligato topologies:
 - eth-1drcl2bdbasemaclrn-eth-2memif-1drcl2xc-1paral
 - eth-1drcl2bdbasemaclrn-eth-2memif-2drcl2xc-1horiz
 - eth-1drcl2bdbasemaclrn-eth-2memif-4drcl2xc-1horiz
 - eth-1drcl2bdbasemaclrn-eth-4memif-2drcl2xc-1chain
 - eth-1drcl2bdbasemaclrn-eth-8memif-4drcl2xc-1chain
 - eth-1drcl2xcbase-eth-2memif-1drcl2xc-1paral
 - eth-1drcl2xcbase-eth-2memif-2drcl2xc-1horiz
 - eth-1drcl2xcbase-eth-2memif-4drcl2xc-1horiz
 - eth-1drcl2xcbase-eth-4memif-2drcl2xc-1chain
 - eth-1drcl2xcbase-eth-8memif-4drcl2xc-1chain

References

2.10.2 Test Code Documentation

[CSIT VPP Performance Tests Documentation](https://docs.fd.io/csit/rls1807/doc/tests.vpp.perf.html)¹⁰⁸ contains detailed functional description and input parameters for each test case.

¹⁰⁸ <https://docs.fd.io/csit/rls1807/doc/tests.vpp.perf.html>

3.1 Overview

DPDK 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 DPDK performance tests please refer to *Physical Testbeds* (page 4).

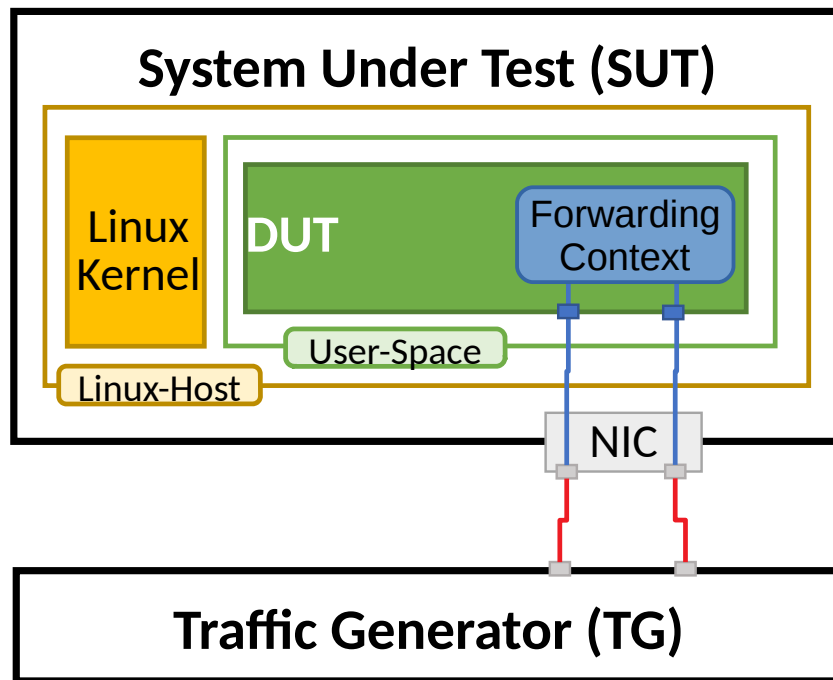
3.1.1 Logical Topologies

CSIT DPDK performance tests are executed on physical testbeds described in *Physical Testbeds* (page 4). Based on the packet path through server SUTs, one distinct logical topology type is used for DPDK DUT data plane testing: NIC-to-NIC switching topology.

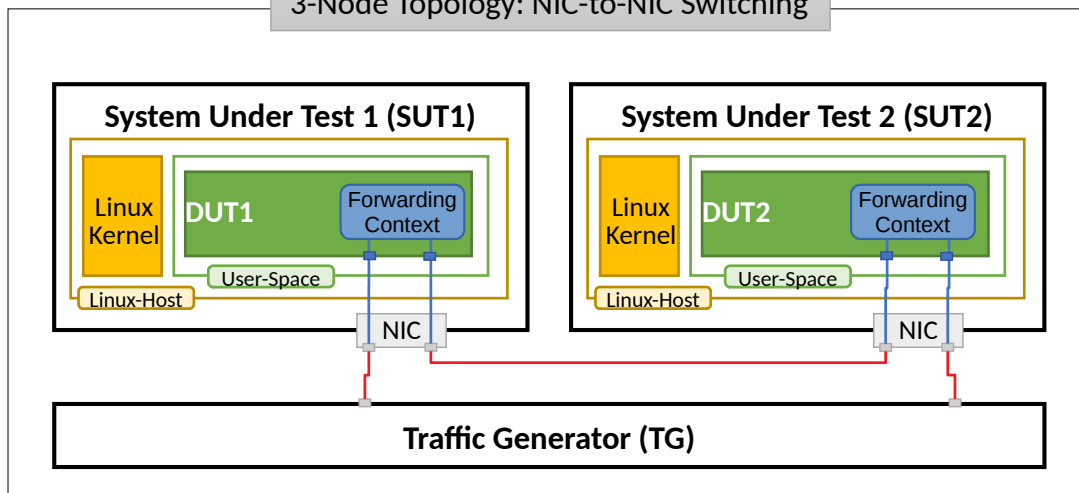
NIC-to-NIC Switching

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

2-Node Topology: NIC-to-NIC Switching



3-Node Topology: NIC-to-NIC Switching



Server Systems Under Test (SUT) run DPDK Testpmd or L3fwd 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 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 limitations. 100GE NICs are not supported in PCIe Gen3 x8 slots.

Note that reported DPDK 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 DPDK packet throughput 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.

3.1.2 Performance Tests Coverage

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

- Packet Throughput: measured in accordance with **RFC 2544**¹⁰⁹, 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): measured 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-18.07 includes following DPDK Testpmd and L3fwd data plane functionality performance tested across a range of NIC drivers and NIC models:

Functionality	Description
L2IntLoop	L2 Interface Loop forwarding all Ethernet frames between two Interfaces.
IPv4 Routed Forwarding	Longest Prefix Match (LPM) L3 IPv4 forwarding of Ethernet frames between two Interfaces, with two /8 prefixes in lookup table.

3.2 Release Notes

3.2.1 Changes in CSIT-18.07

1. DPDK RELEASE VERSION CHANGE

- CSIT-18.07 tested DPDK 18.05, as used by VPP-18.07 release.

2. DPDK PERFORMANCE TESTS

- **MRR Throughput:** MRR (Maximum Receive Rate) test code has now configurable trial duration and number of consecutive executions. Coverage of MRR tests has been extended across more test scenarios. MRR tests are used for continuous performance trending.
- **MLRsearch Throughput:** MLRsearch algorithm has been introduced for all NDR and PDR throughput tests. All tests that previously used binary search got converted to MLRsearch. Coverage of NDR/PDR tests has been extended across more test scenarios.
- **2-Node Tests:** A new set of 2-node tests covering DPDK Testpmd and L3fwd, running on new Xeon Skylake testbeds.
- **More NICs:** Increased coverage of NIC models tested: Intel-xxv710-da2, Intel-x710-da4.

¹⁰⁹ <https://tools.ietf.org/html/rfc2544.html>

- **Generated tests:** Simplified and unified test structure, semi- autogenerated by generator script. Test generator is currently able to create test combinations with various frame size and cores combinations. All existing test cases were converted to new format.

3. PRESENTATION AND ANALYTICS LAYER

- **Performance trending:** Added regular performance trending, anomaly detection and analysis for DPDK.

3.2.2 Known Issues

No known issues.

3.3 Packet Throughput

Throughput graphs are generated by multiple executions of the same performance tests across physical testbeds hosted LF FD.io labs: 3n-hsw, 2n-skx, 2n-skx. Box-and-Whisker plots are used to display variations in measured throughput values, without making any assumptions of the underlying statistical distribution.

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 DPDK 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.

Note: Test results have been generated by [FD.io test executor dpdk performance job 3n-hsw¹¹⁰](#), [FD.io test executor dpdk performance job 3n-skx¹¹¹](#) and [FD.io test executor dpdk performance job 2n-skx¹¹²](#) with RF result files csit-dpdk-perf-1807-*.zip [archived here](#). Required per test case data set size is **10** and for DPDK tests this is the actual size, as all scheduled test executions completed successfully.

¹¹⁰ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-hsw>

¹¹¹ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-skx>

¹¹² <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-2n-skx>

3.3.1 Testpmd

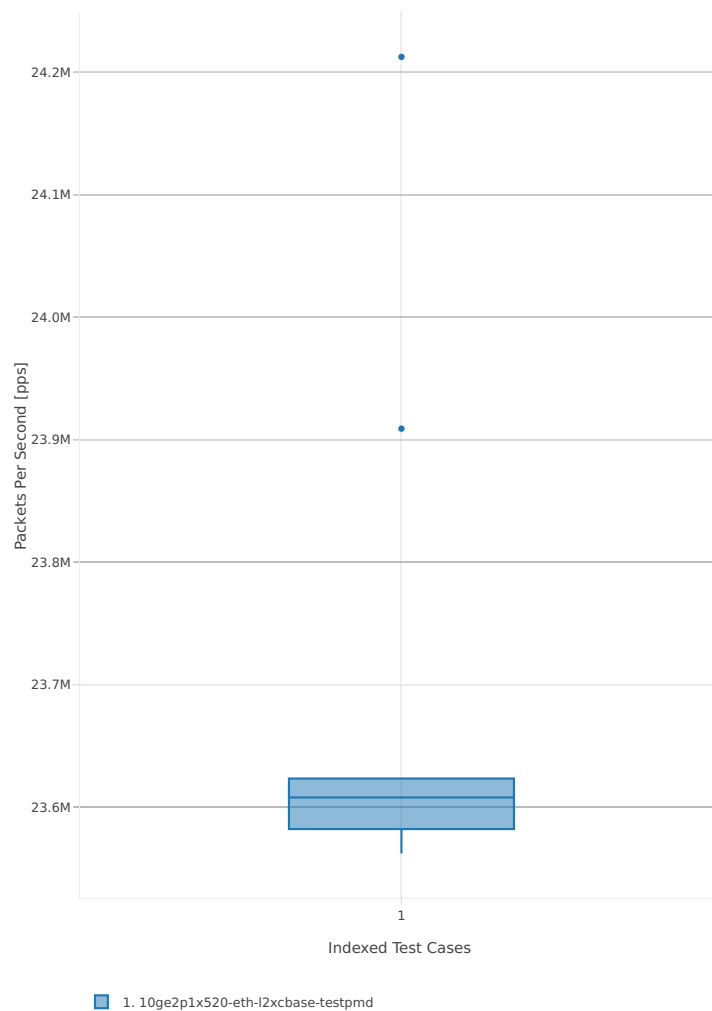
Following sections include summary graphs of DPDK Testpmd Phy-to-Phy performance with L2 Ethernet Interface Loop, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for Testpmd running in multiple configurations of Testpmd pmd thread(s), a.k.a. Testpmd 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](#)¹¹³.

3n-hsw-x520

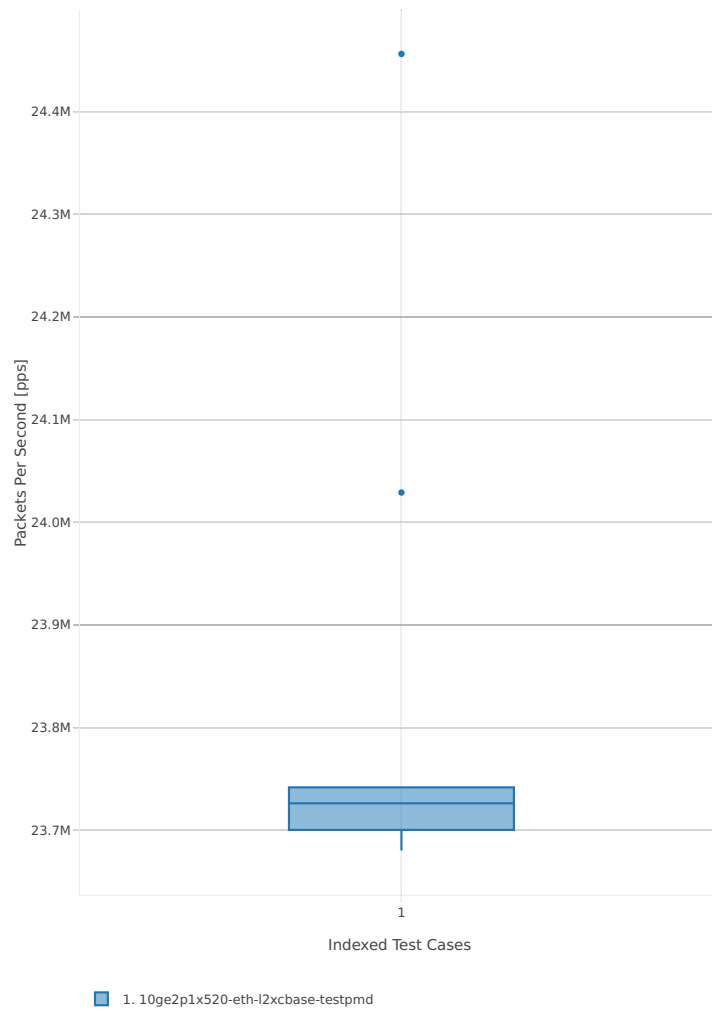
64b-1t1c-base

Throughput: testpmd-3n-hsw-x520-64b-1t1c-base-ndr



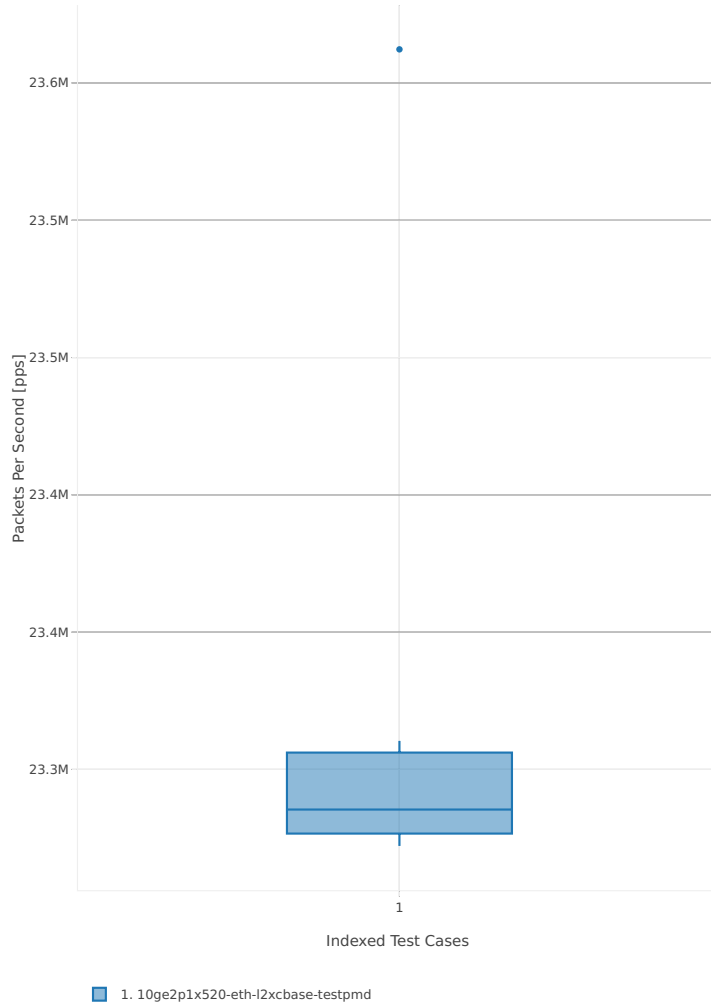
¹¹³ <https://git.fd.io/csit/tree/tests/dpdk/perf?h=rls1807>

Throughput: testpmd-3n-hsw-x520-64b-1t1c-base-pdr

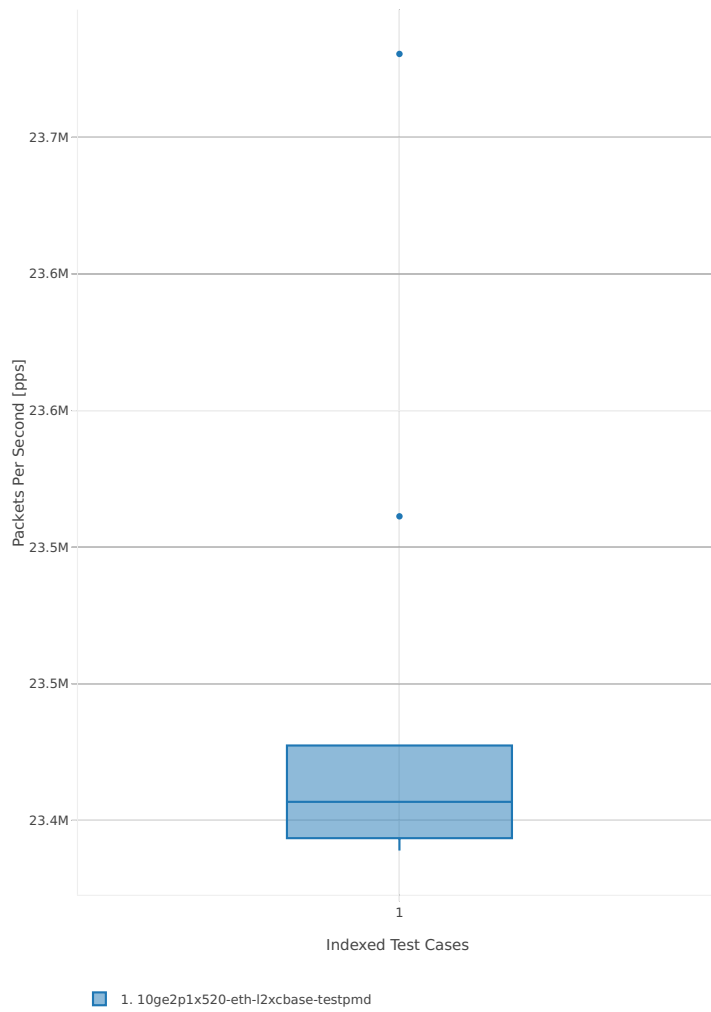


64b-2t2c-base

Throughput: testpmd-3n-hsw-x520-64b-2t2c-base-ndr



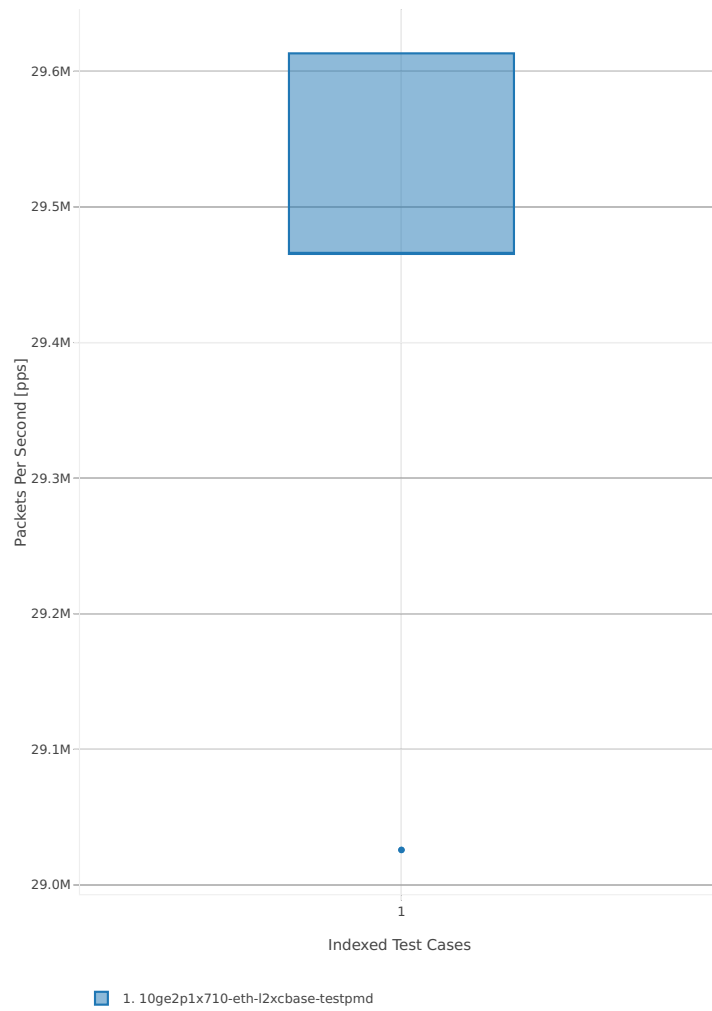
Throughput: testpmd-3n-hsw-x520-64b-2t2c-base-pdr



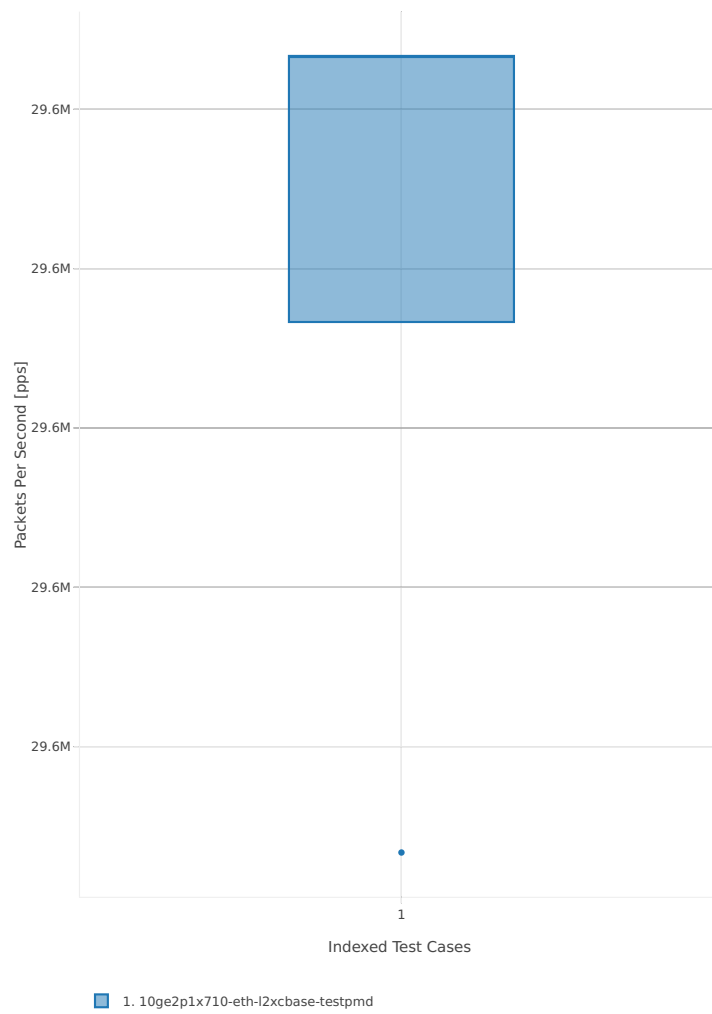
3n-hsw-x710

64b-1t1c-base

Throughput: testpmd-3n-hsw-x710-64b-1t1c-base-ndr

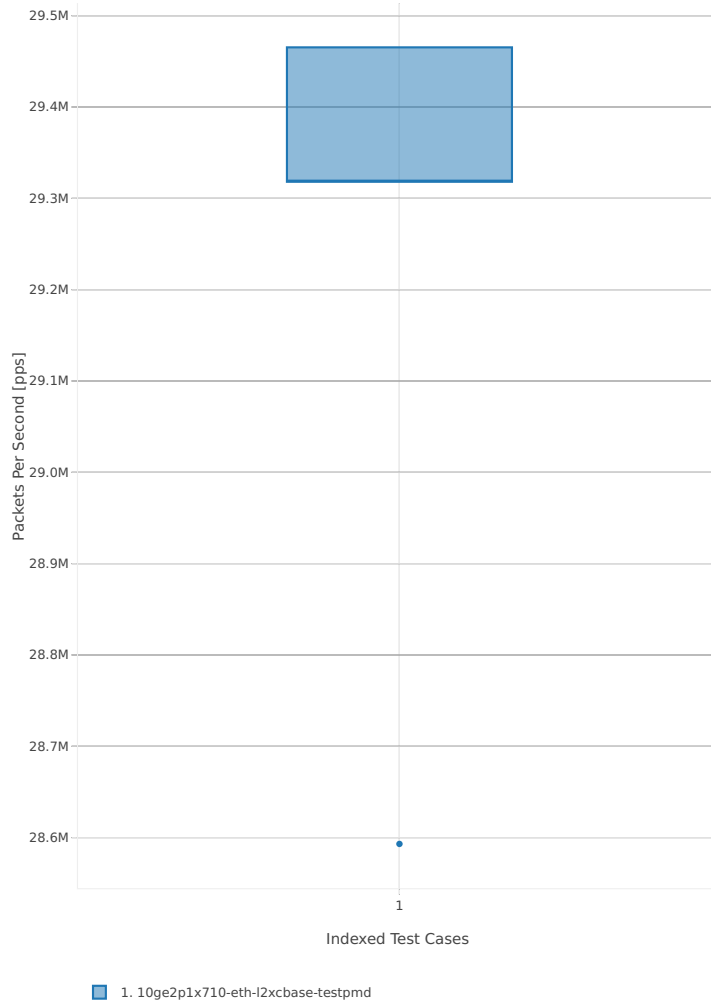


Throughput: testpmd-3n-hsw-x710-64b-1t1c-base-pdr

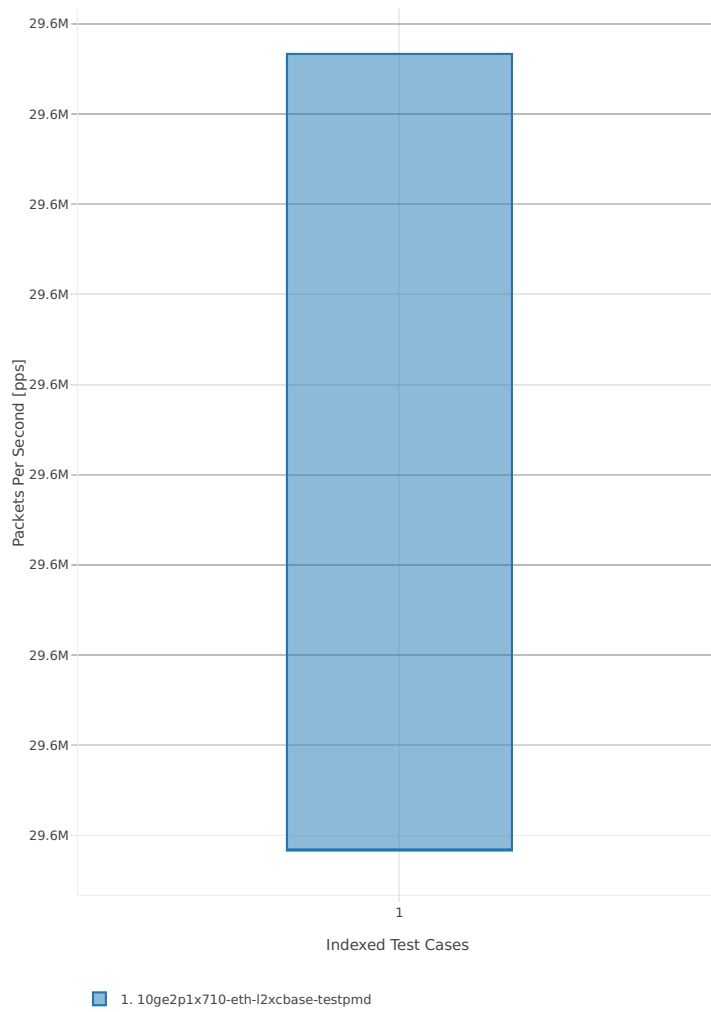


64b-2t2c-base

Throughput: testpmd-3n-hsw-x710-64b-2t2c-base-ndr

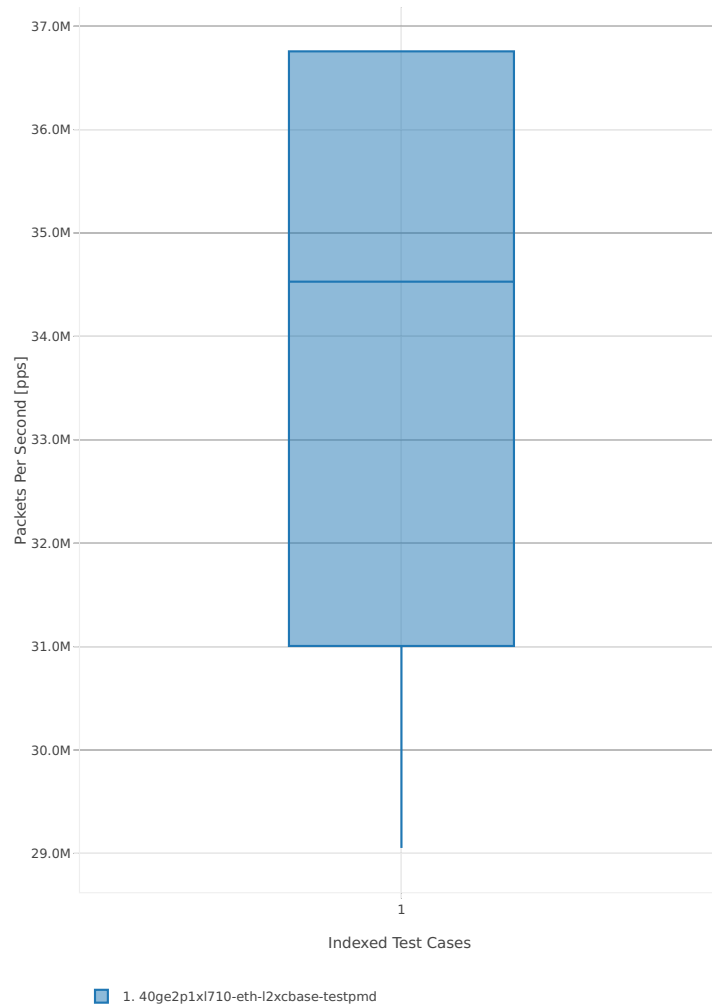


Throughput: testpmd-3n-hsw-x710-64b-2t2c-base-pdr

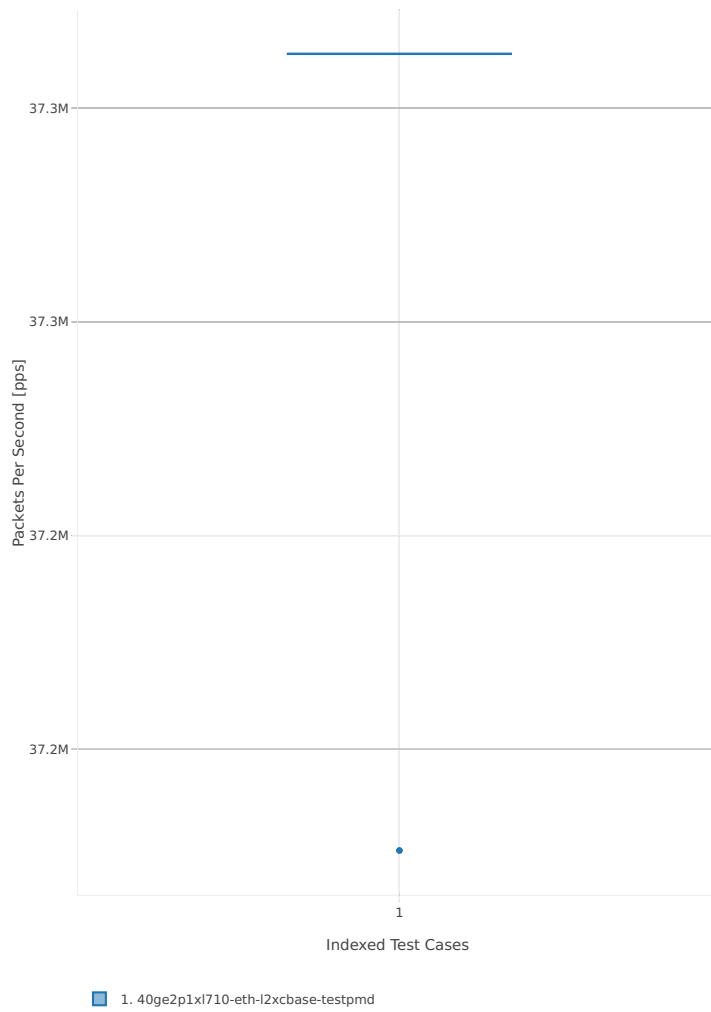


3n-hsw-xl710**64b-1t1c-base**

Throughput: testpmd-3n-hsw-xl710-64b-1t1c-base-ndr

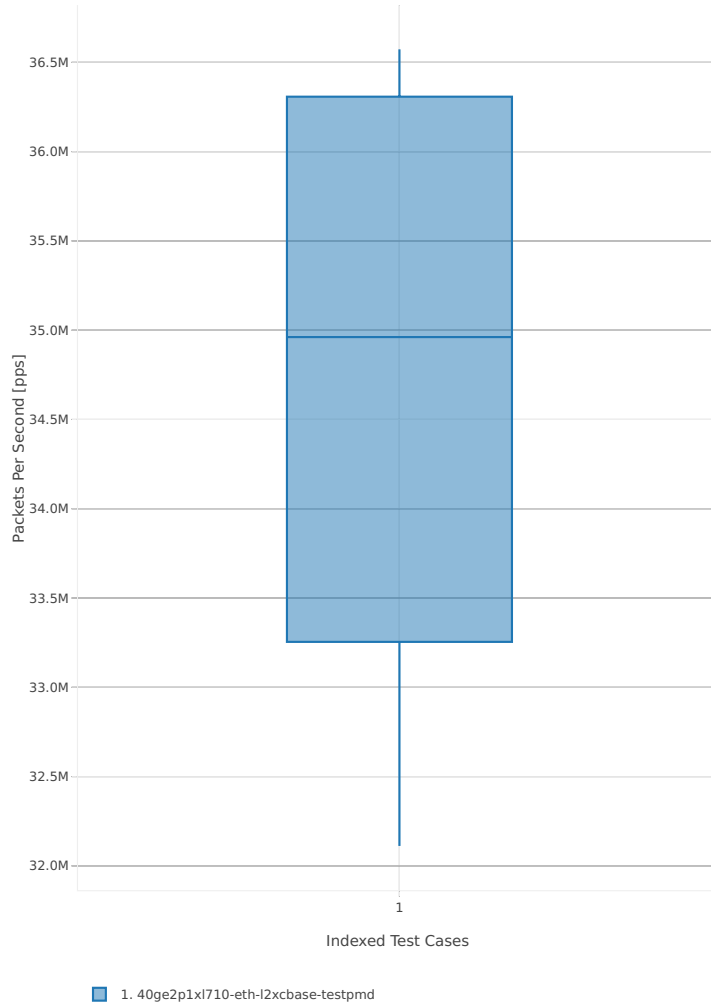


Throughput: testpmd-3n-hsw-xl710-64b-1t1c-base-pdr

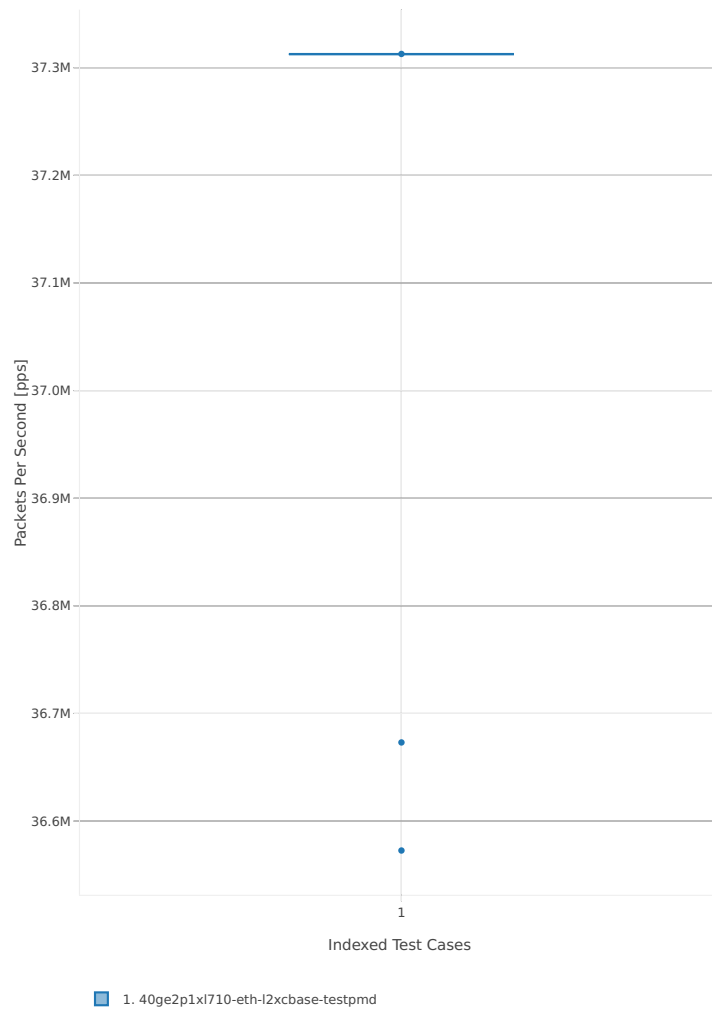


64b-2t2c-base

Throughput: testpmd-3n-hsw-xl710-64b-2t2c-base-ndr



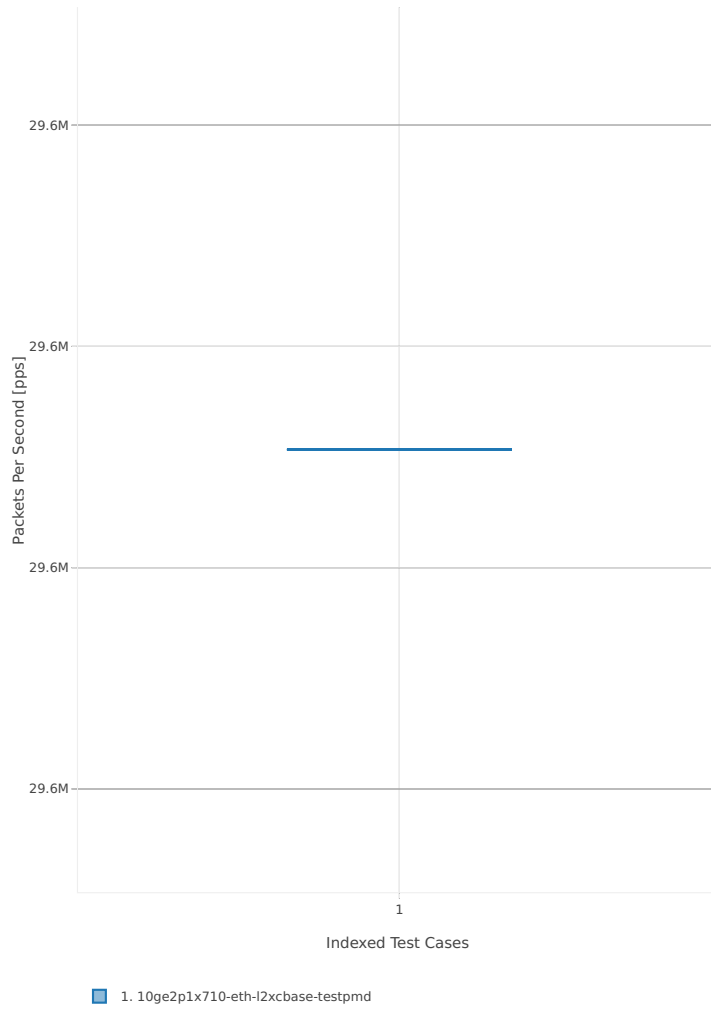
Throughput: testpmd-3n-hsw-xl710-64b-2t2c-base-pdr



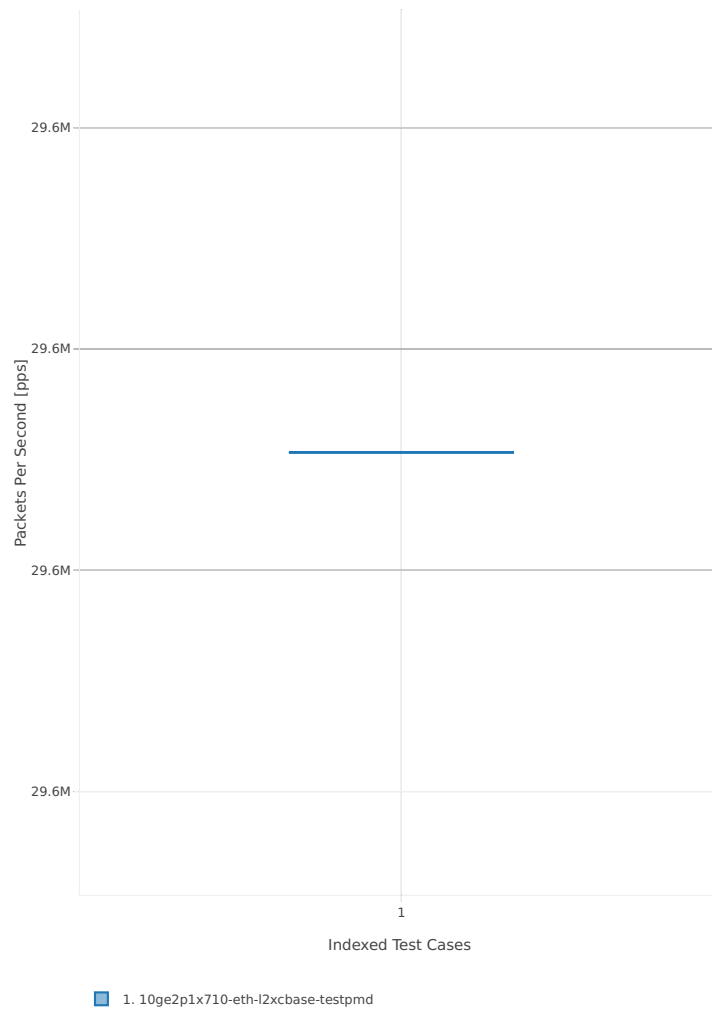
3n-skx-x710

64b-2t1c-base

Throughput: testpmd-3n-skx-x710-64b-2t1c-base-ndr

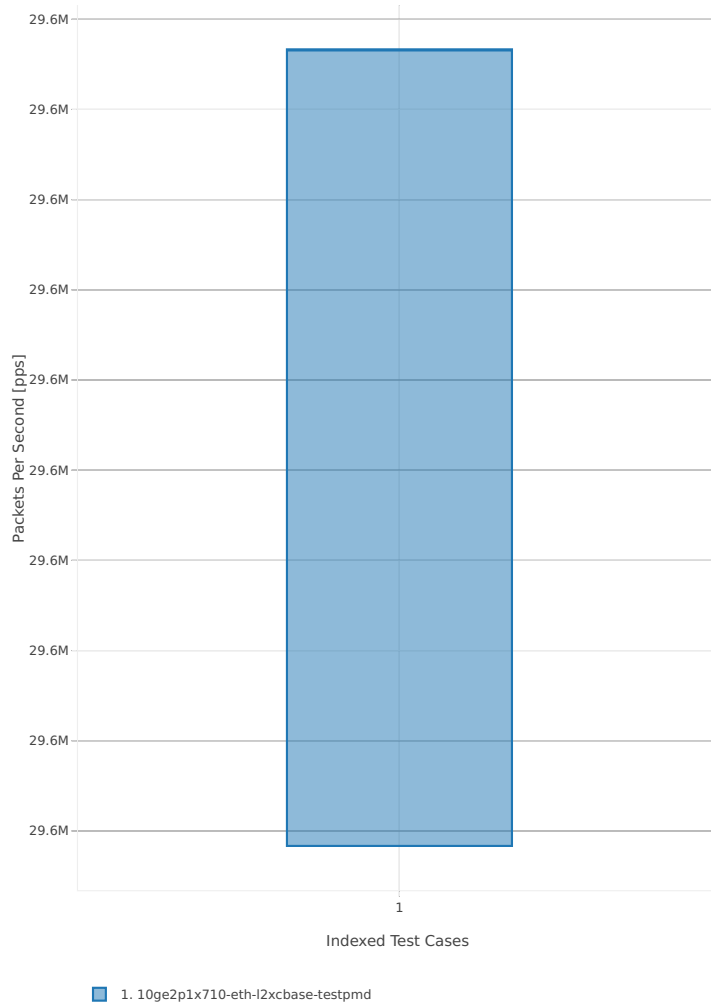


Throughput: testpmd-3n-skx-x710-64b-2t1c-base-pdr

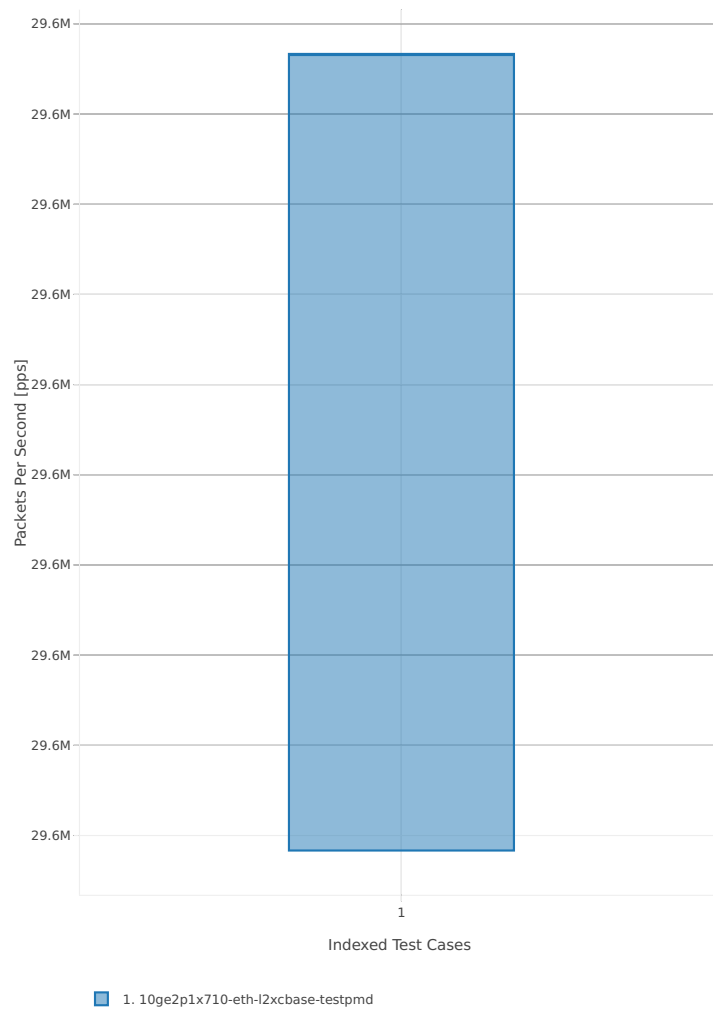


64b-4t2c-base

Throughput: testpmd-3n-skx-x710-64b-4t2c-base-ndr



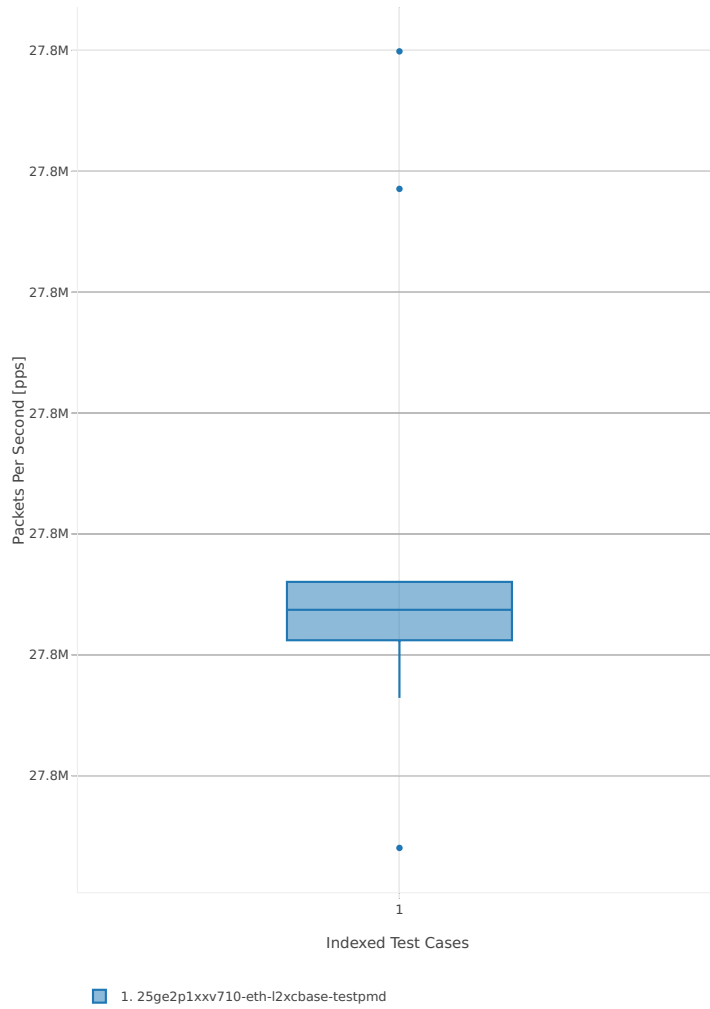
Throughput: testpmd-3n-skx-x710-64b-4t2c-base-pdr



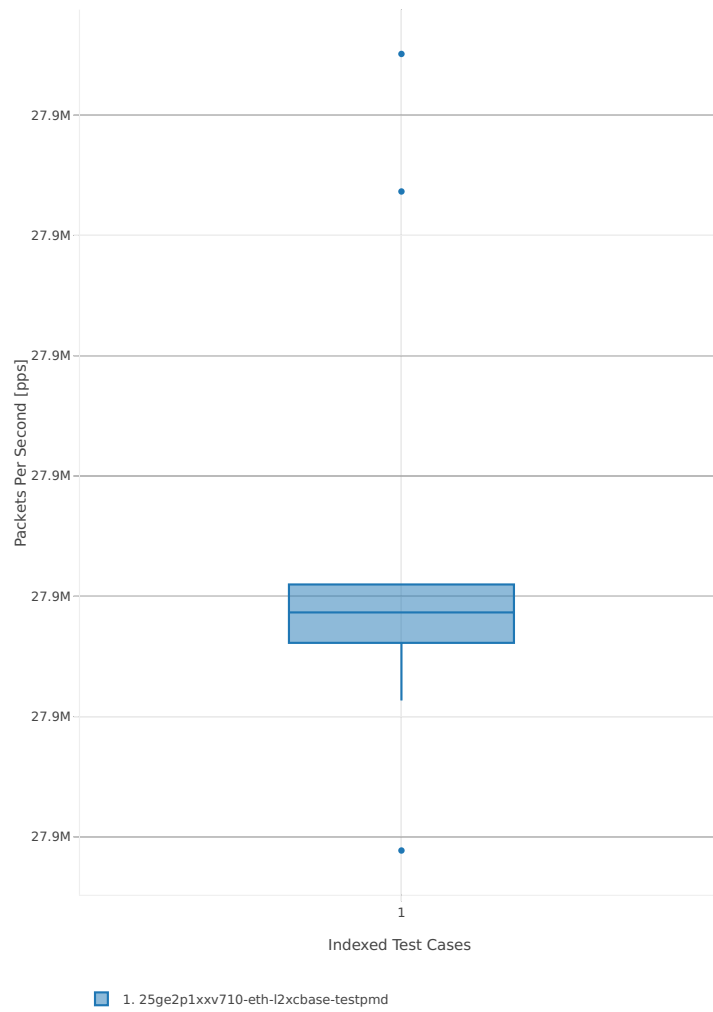
3n-skx-xxv710

64b-2t1c-base

Throughput: testpmd-3n-skx-xxv710-64b-2t1c-base-ndr

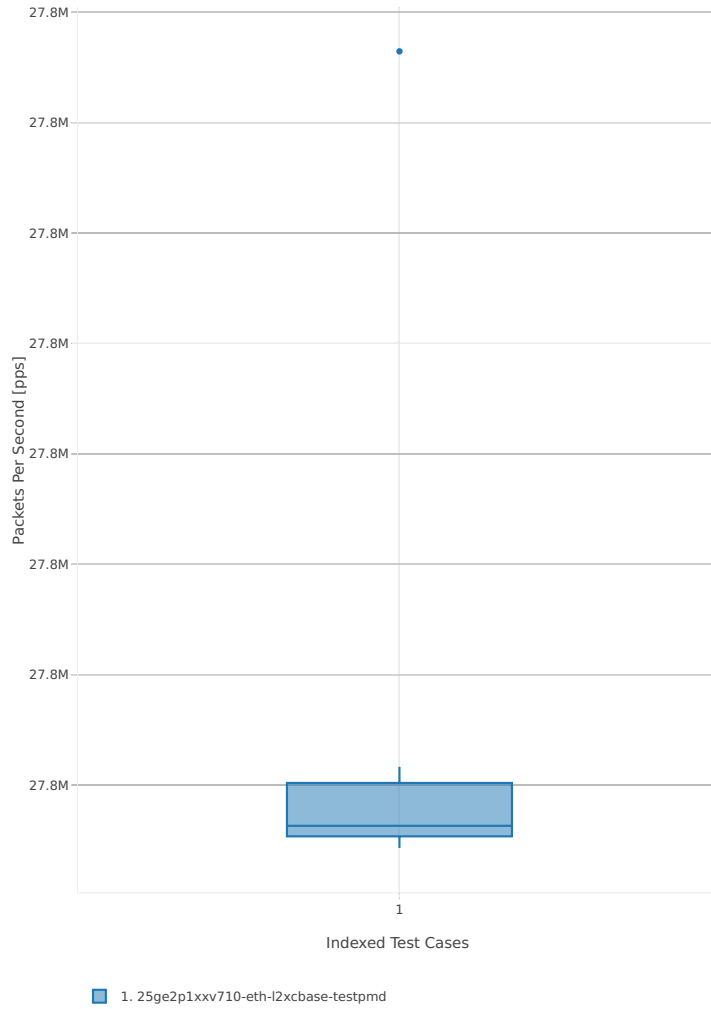


Throughput: testpmd-3n-skx-xxv710-64b-2t1c-base-pdr

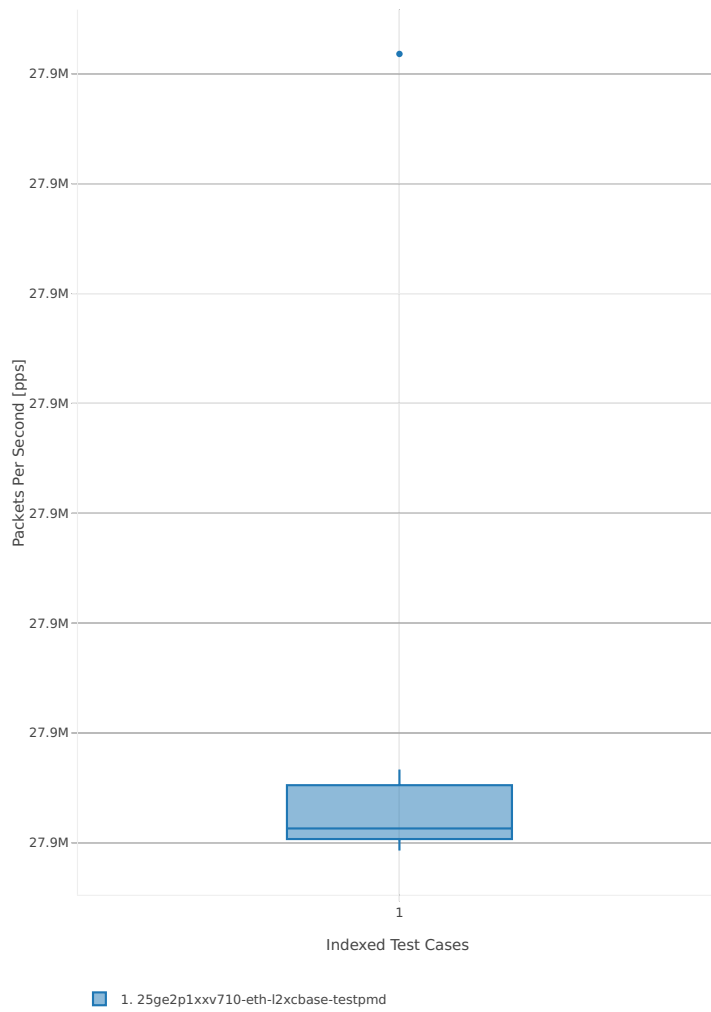


64b-4t2c-base

Throughput: testpmd-3n-skx-xxv710-64b-4t2c-base-ndr



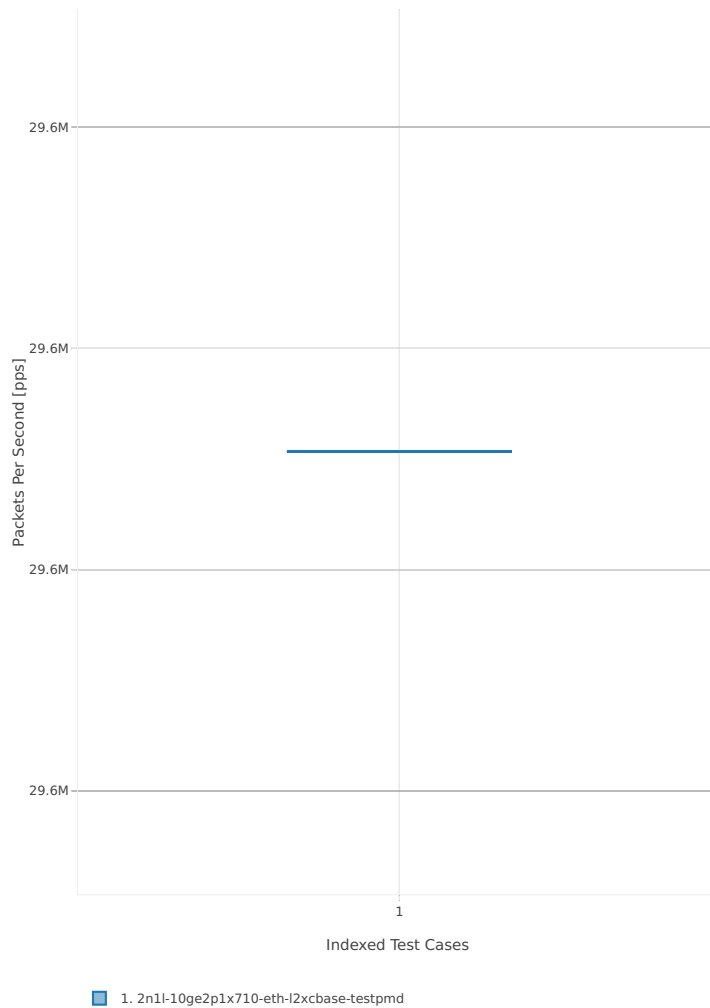
Throughput: testpmd-3n-skx-xxv710-64b-4t2c-base-pdr



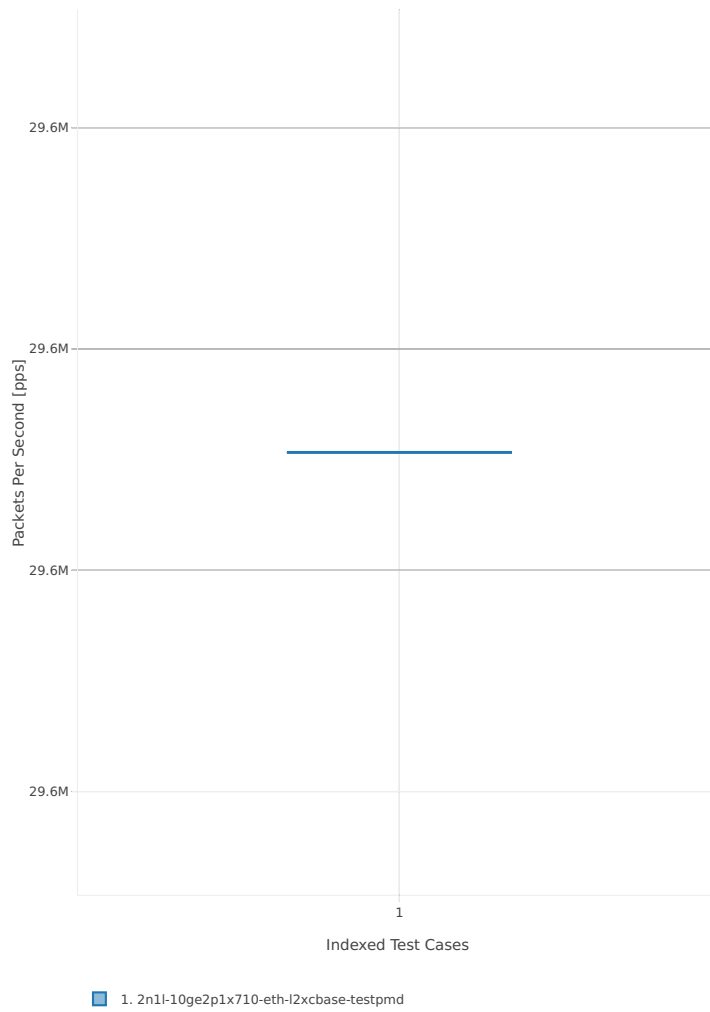
2n-skx-x710

64b-2t1c-base

Throughput: testpmd-2n-skx-x710-64b-2t1c-base-ndr

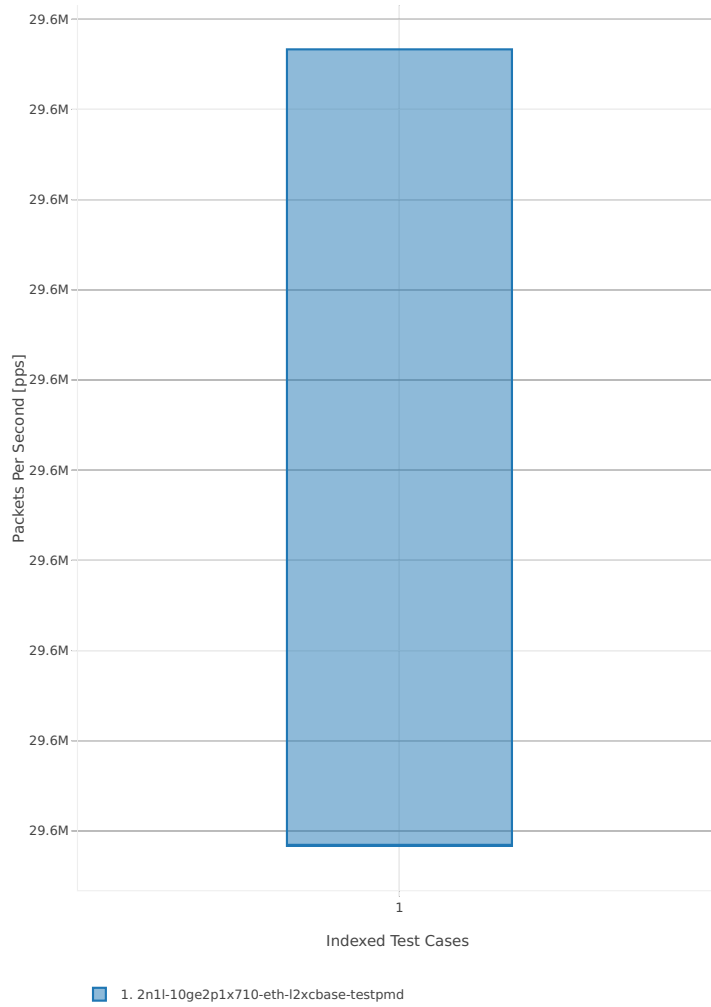


Throughput: testpmd-2n-skx-x710-64b-2t1c-base-pdr

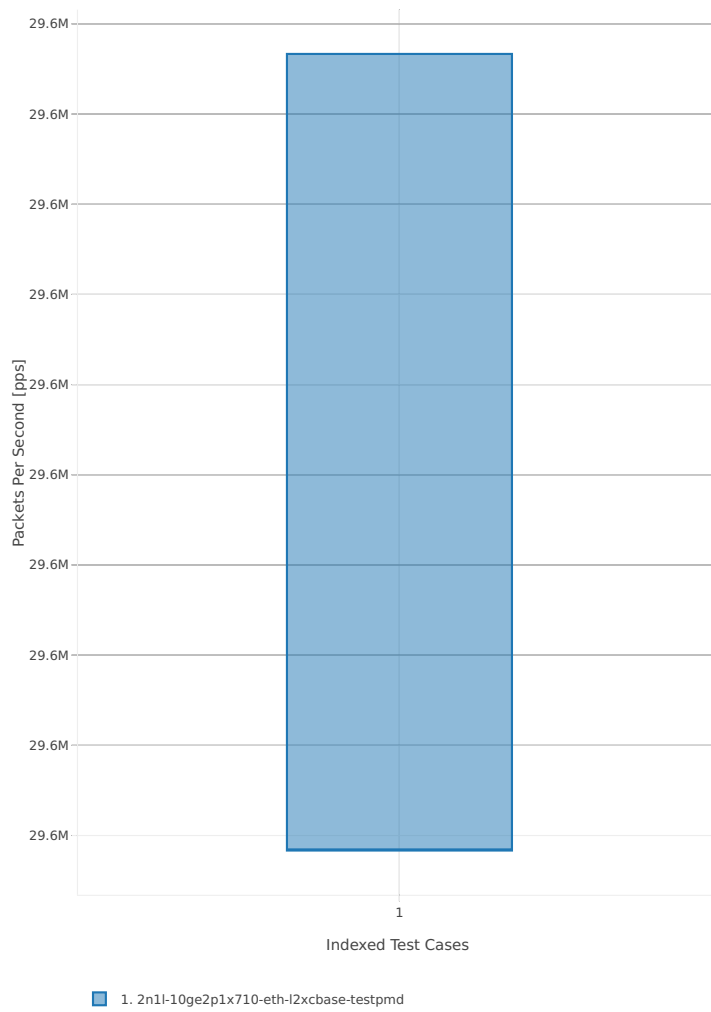


64b-4t2c-base

Throughput: testpmd-2n-skx-x710-64b-4t2c-base-ndr



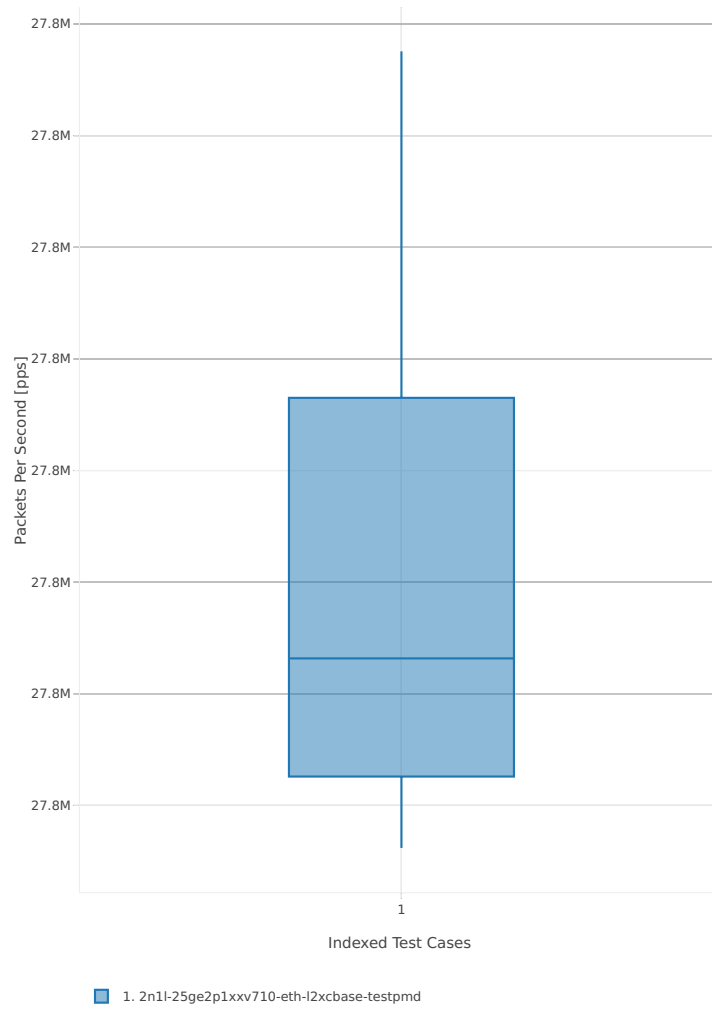
Throughput: testpmd-2n-skx-x710-64b-4t2c-base-pdr



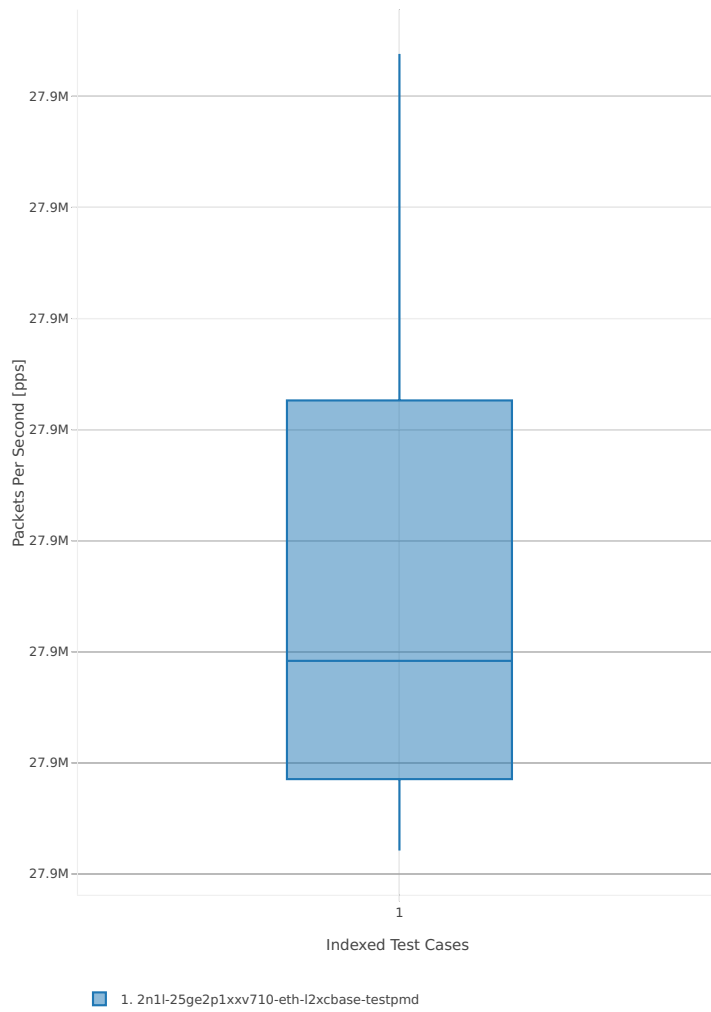
2n-skx-xxv710

64b-2t1c-base

Throughput: testpmd-2n-skx-xxv710-64b-2t1c-base-ndr

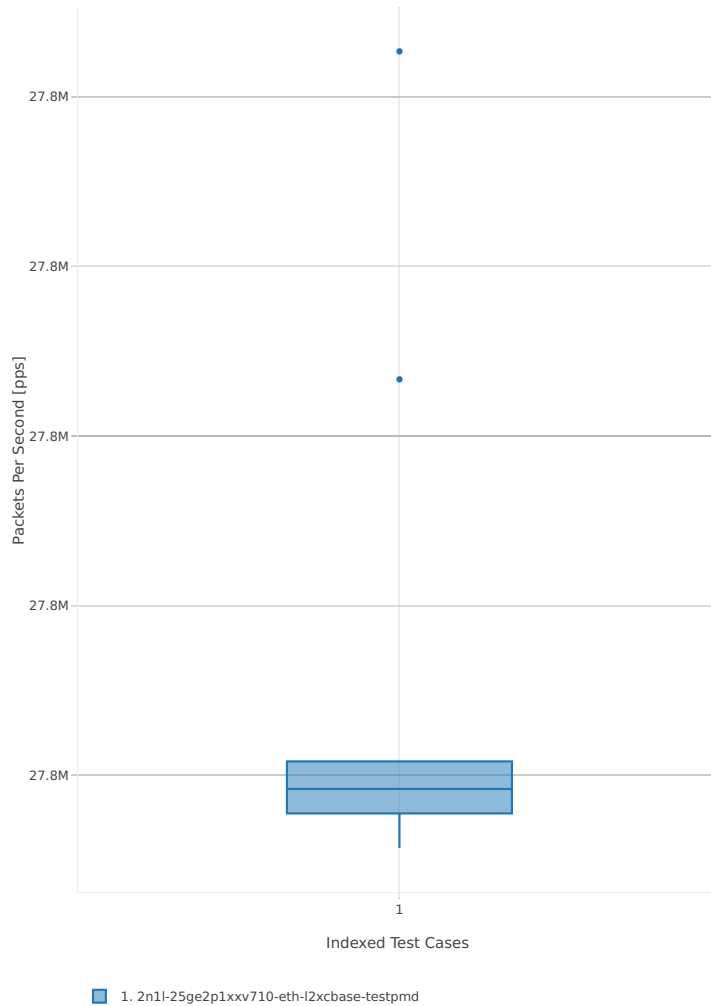


Throughput: testpmd-2n-skx-xxv710-64b-2t1c-base-pdr

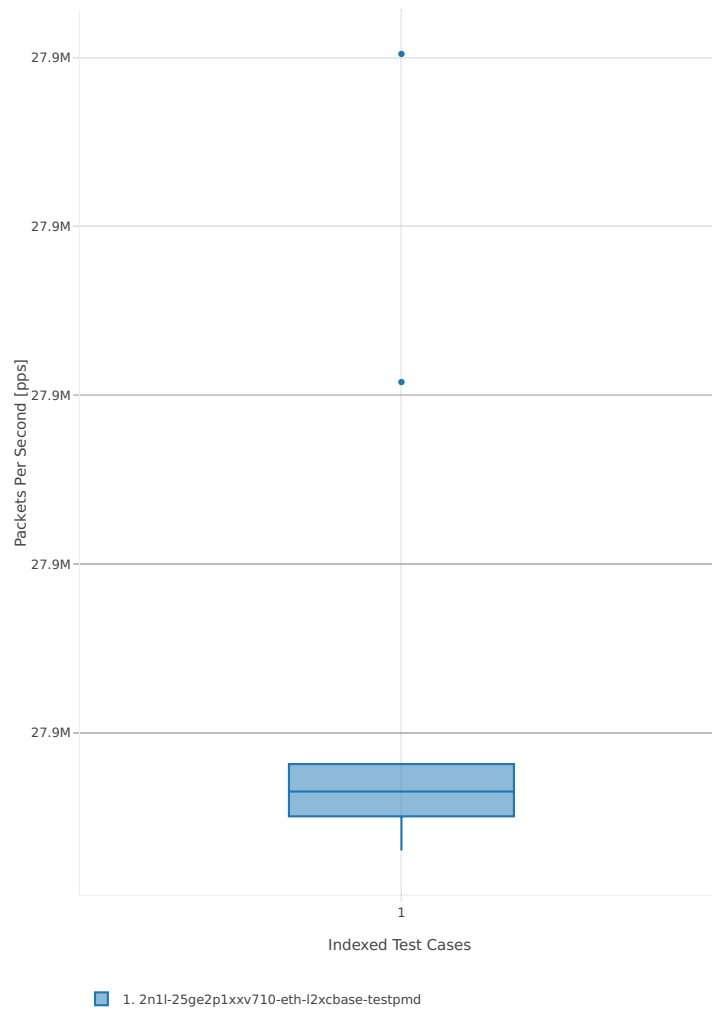


64b-4t2c-base

Throughput: testpmd-2n-skx-xxv710-64b-4t2c-base-ndr



Throughput: testpmd-2n-skx-xxv710-64b-4t2c-base-pdr



3.3.2 L3fwd

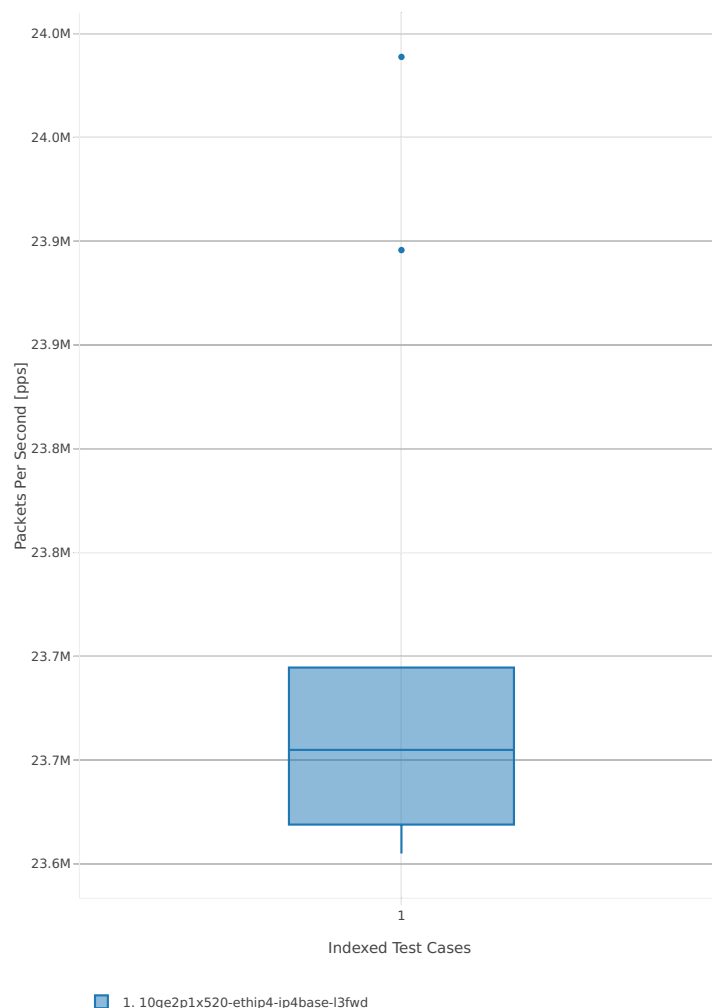
Following sections include summary graphs of L3FWD Phy-to-Phy performance with packet routed forwarding, including NDR throughput (zero packet loss) and PDR throughput (<0.5% packet loss). Performance is reported for L3FWD running in multiple configurations of L3FWD pmd thread(s), a.k.a. L3FWD data plane 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](#)¹¹⁴.

3n-hsw-x520

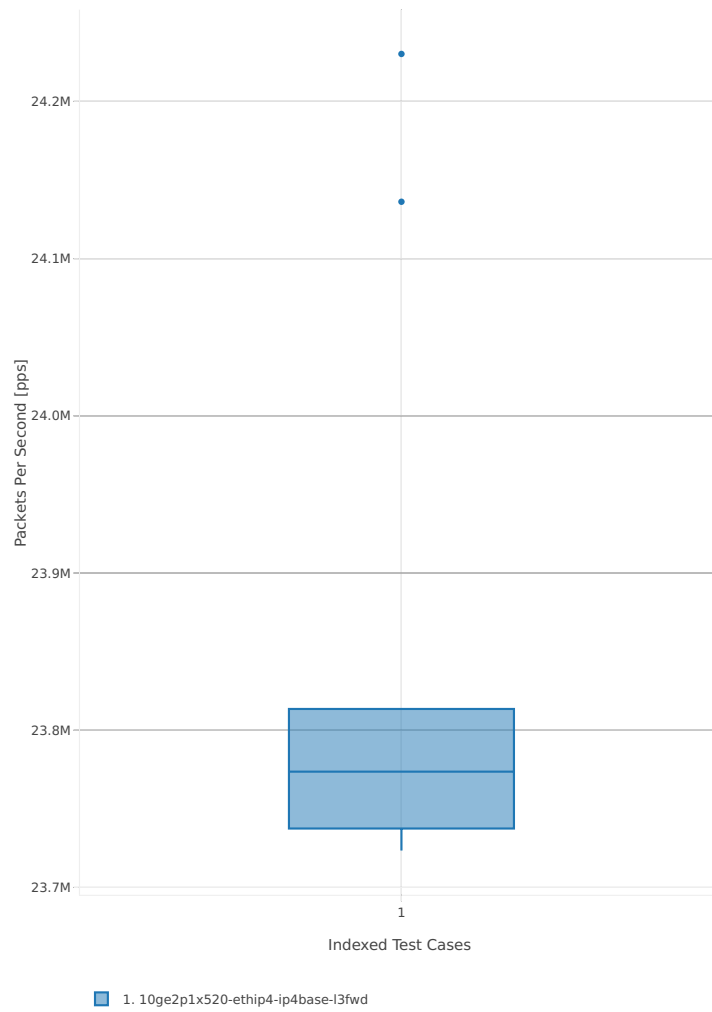
64b-1t1c-base

Throughput: l3fwd-3n-hsw-x520-64b-1t1c-base-ndr



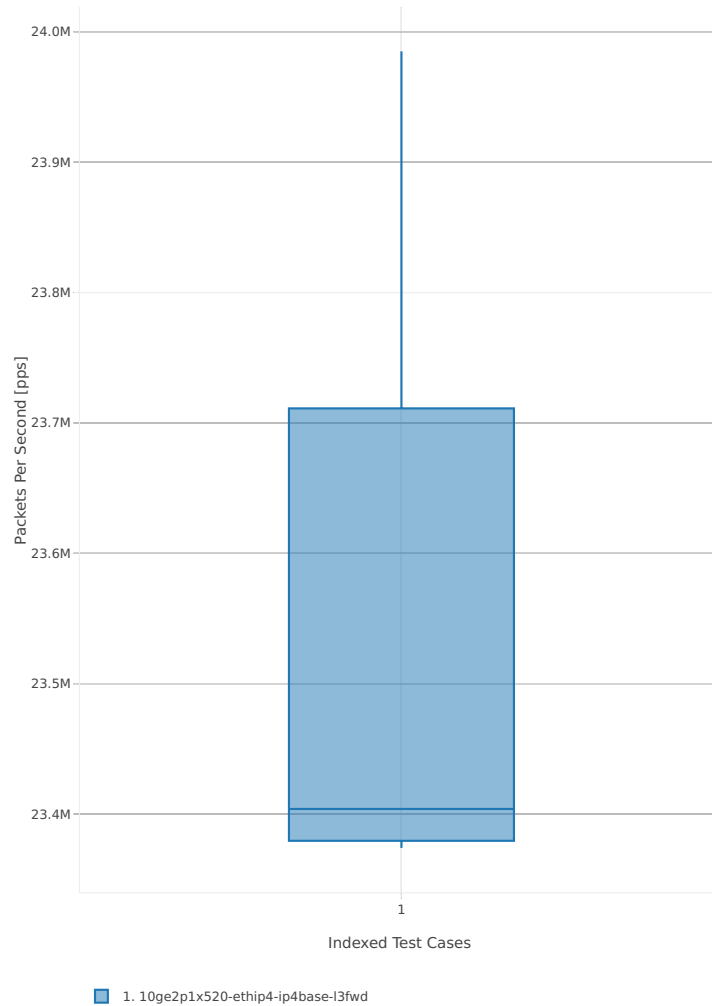
¹¹⁴ <https://git.fd.io/csit/tree/tests/dpdk/perf?h=rls1807>

Throughput: l3fwd-3n-hsw-x520-64b-1t1c-base-pdr

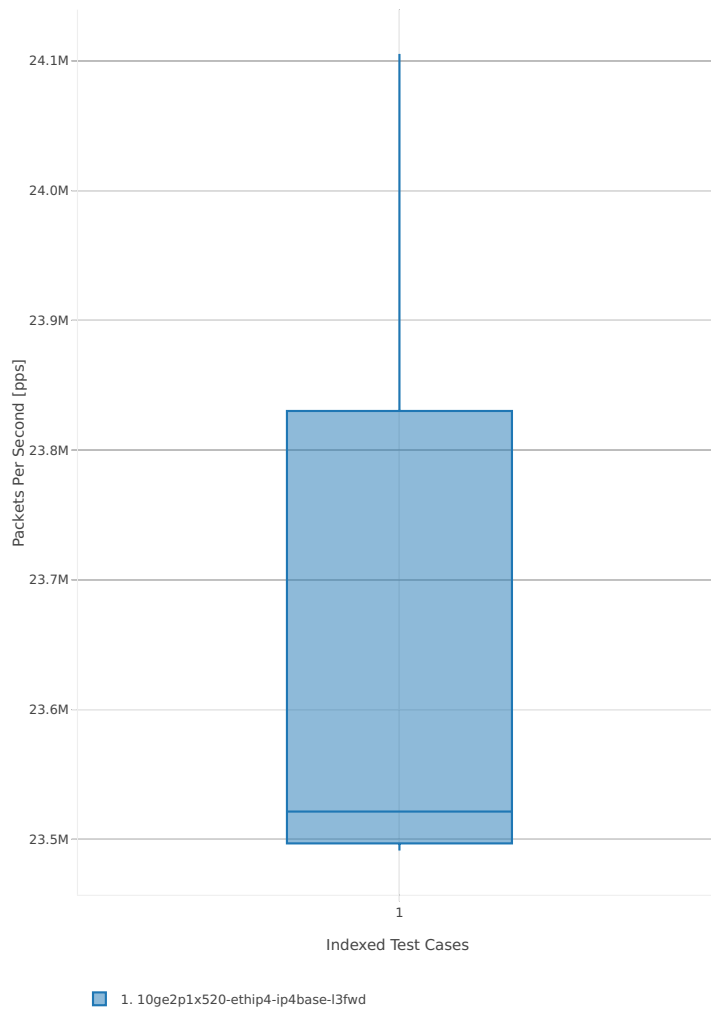


64b-2t2c-base

Throughput: l3fwd-3n-hsw-x520-64b-2t2c-base-ndr



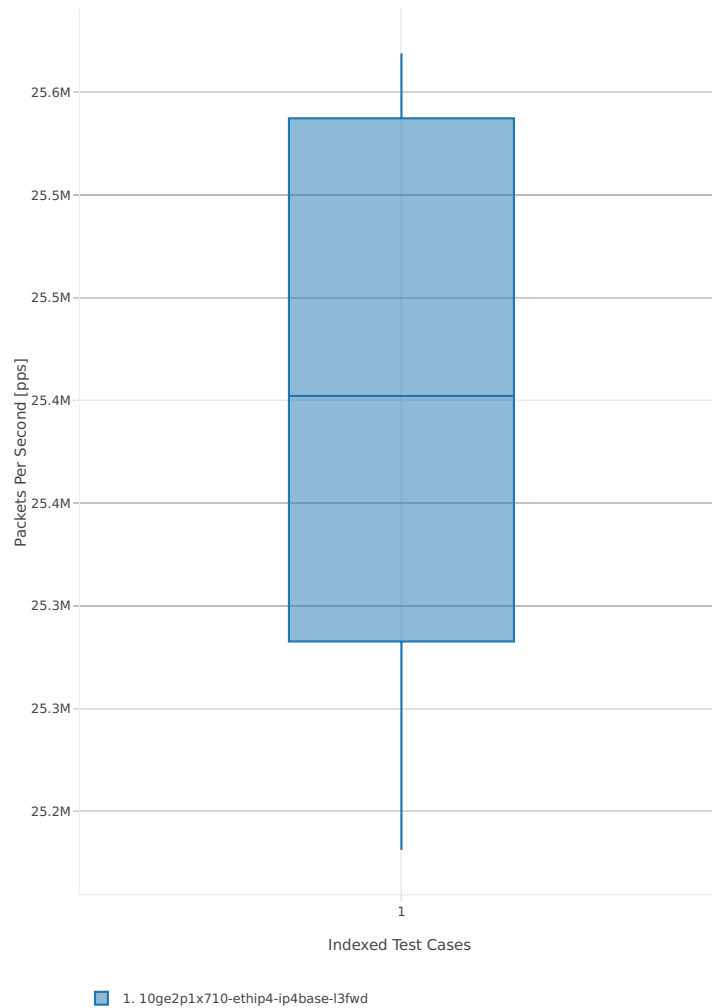
Throughput: l3fwd-3n-hsw-x520-64b-2t2c-base-pdr



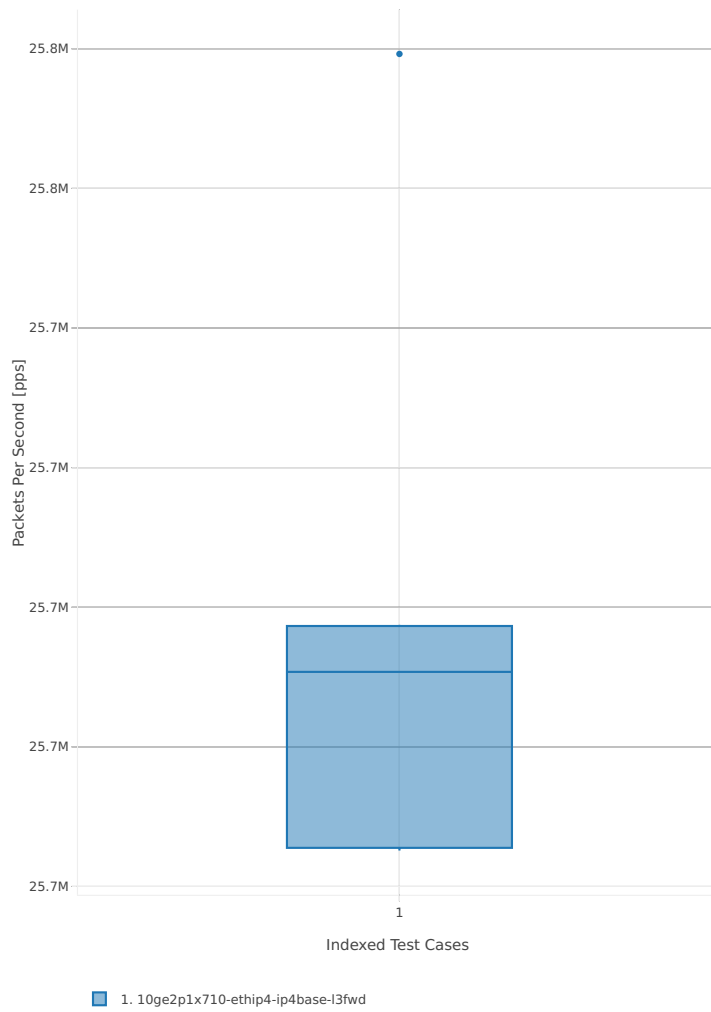
3n-hsw-x710

64b-1t1c-base

Throughput: l3fwd-3n-hsw-x710-64b-1t1c-base-ndr

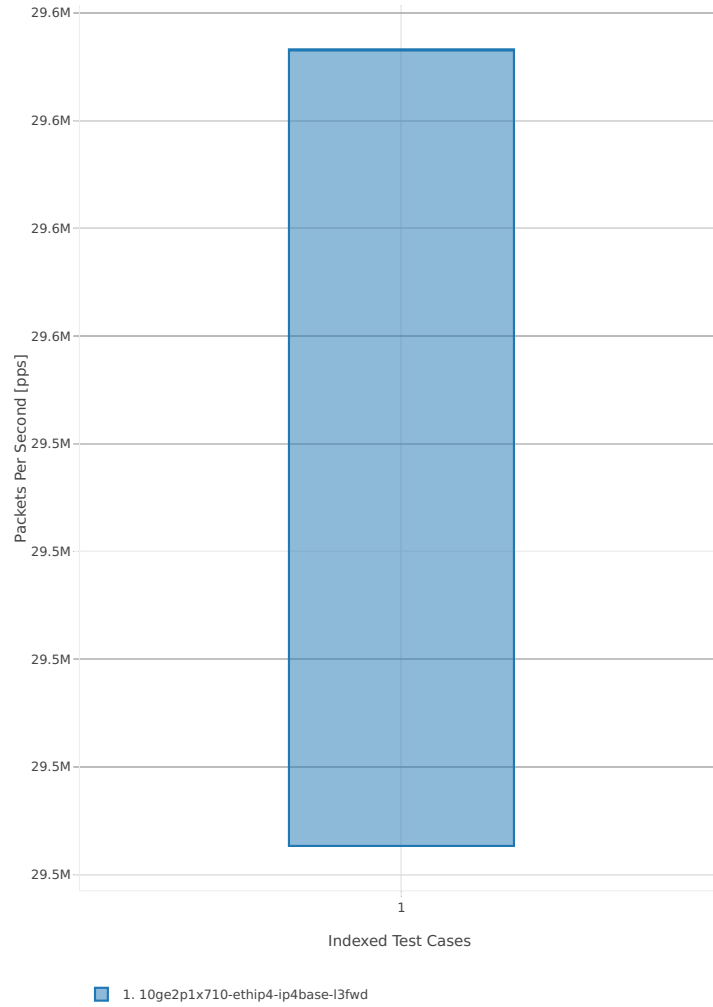


Throughput: l3fwd-3n-hsw-x710-64b-1t1c-base-pdr

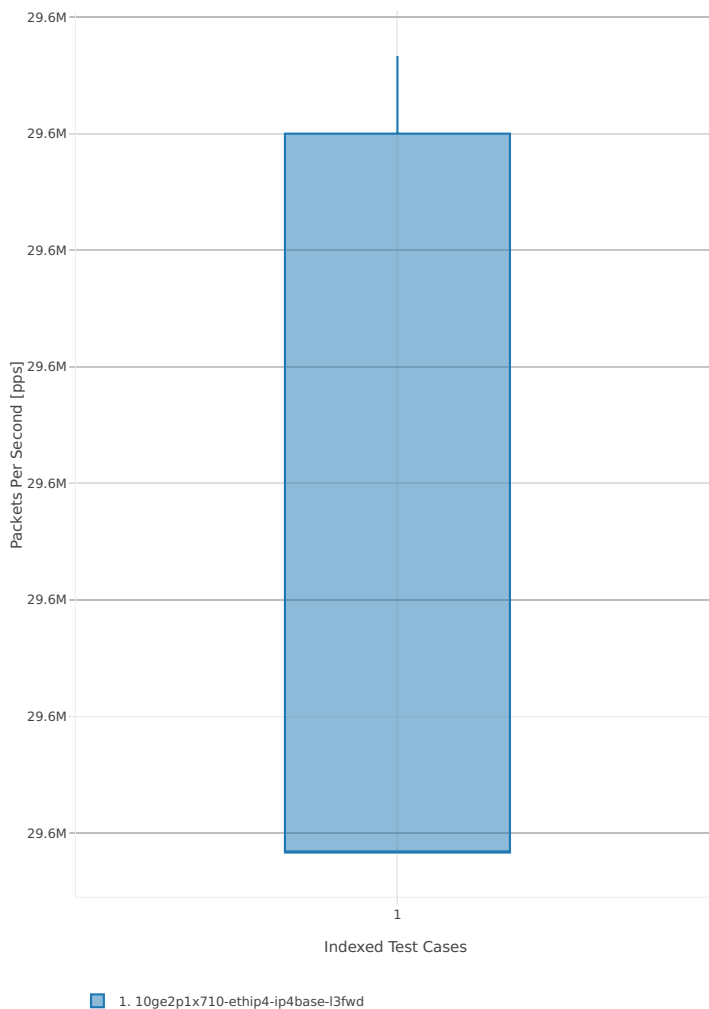


64b-2t2c-base

Throughput: l3fwd-3n-hsw-x710-64b-2t2c-base-ndr



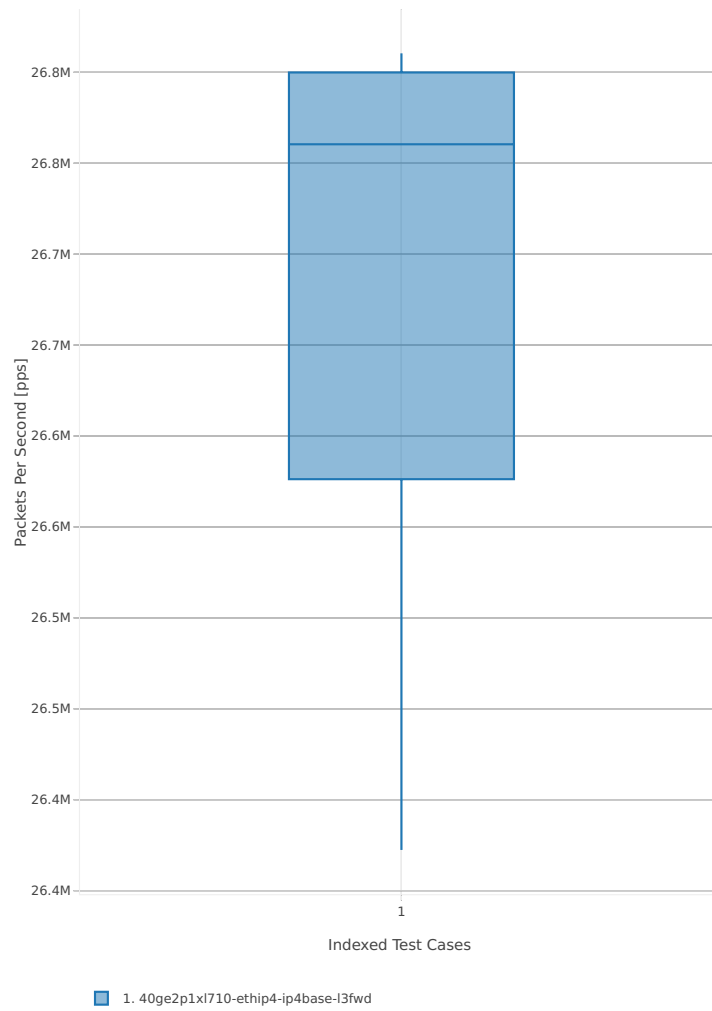
Throughput: l3fwd-3n-hsw-x710-64b-2t2c-base-pdr



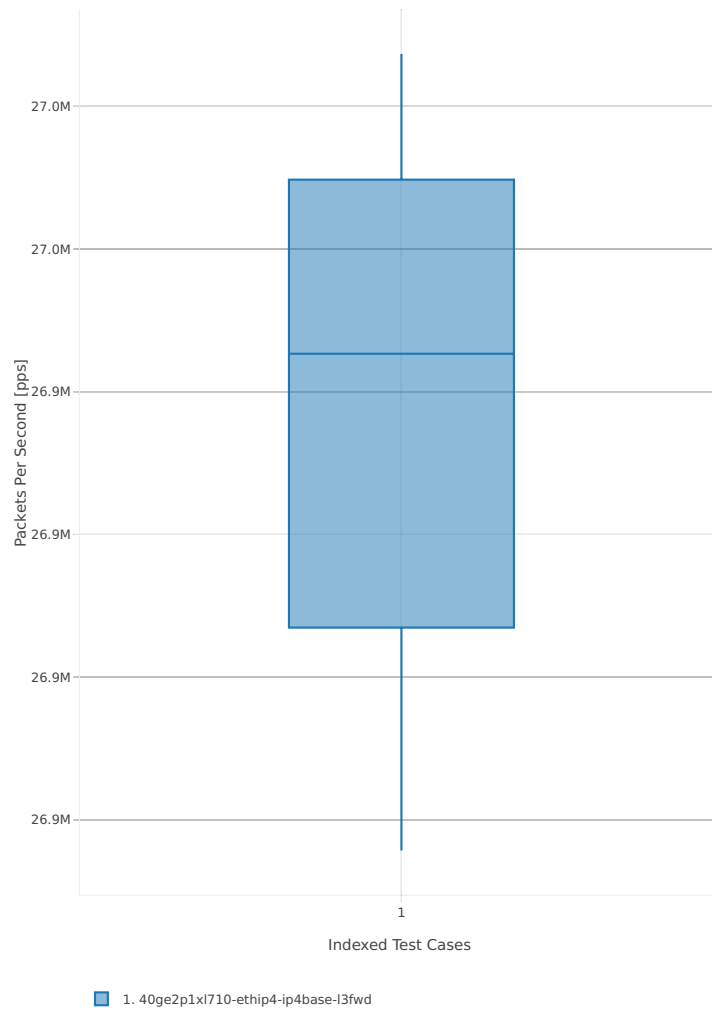
3n-hsw-xl710

64b-1t1c-base

Throughput: I3fwd-3n-hsw-xl710-64b-1t1c-base-ndr

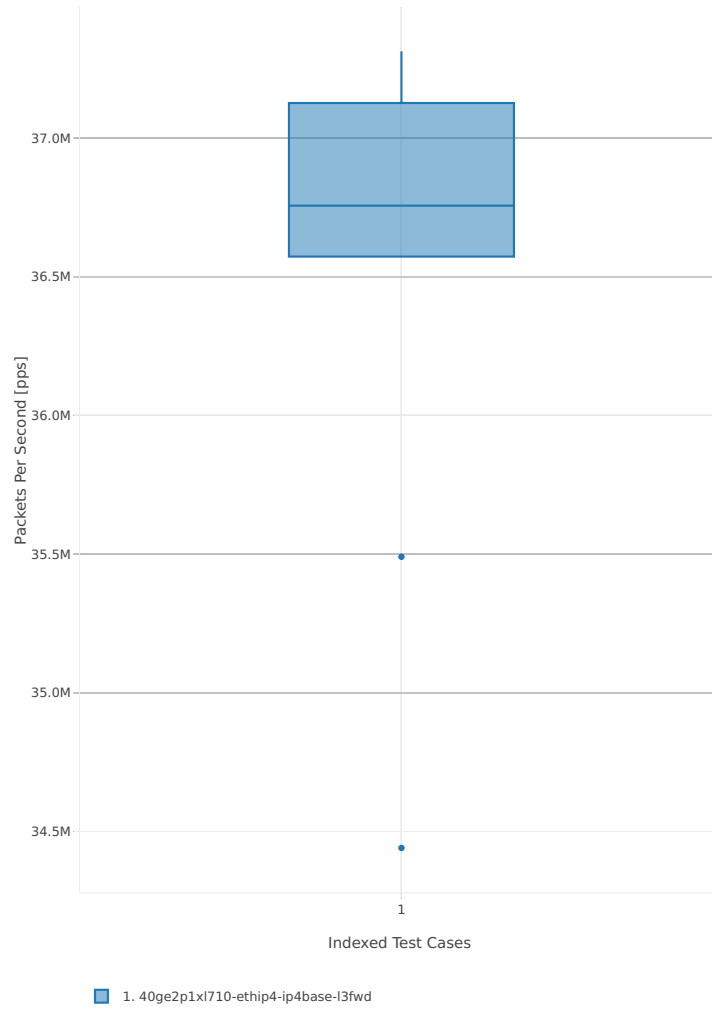


Throughput: l3fwd-3n-hsw-xl710-64b-1t1c-base-pdr

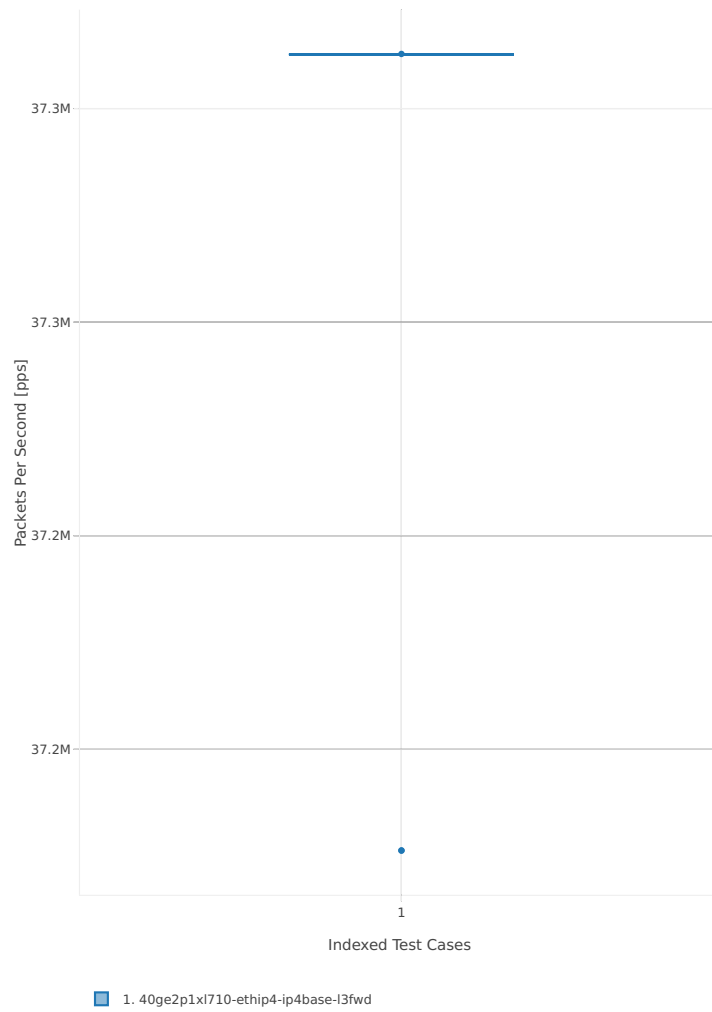


64b-2t2c-base

Throughput: l3fwd-3n-hsw-xl710-64b-2t2c-base-ndr



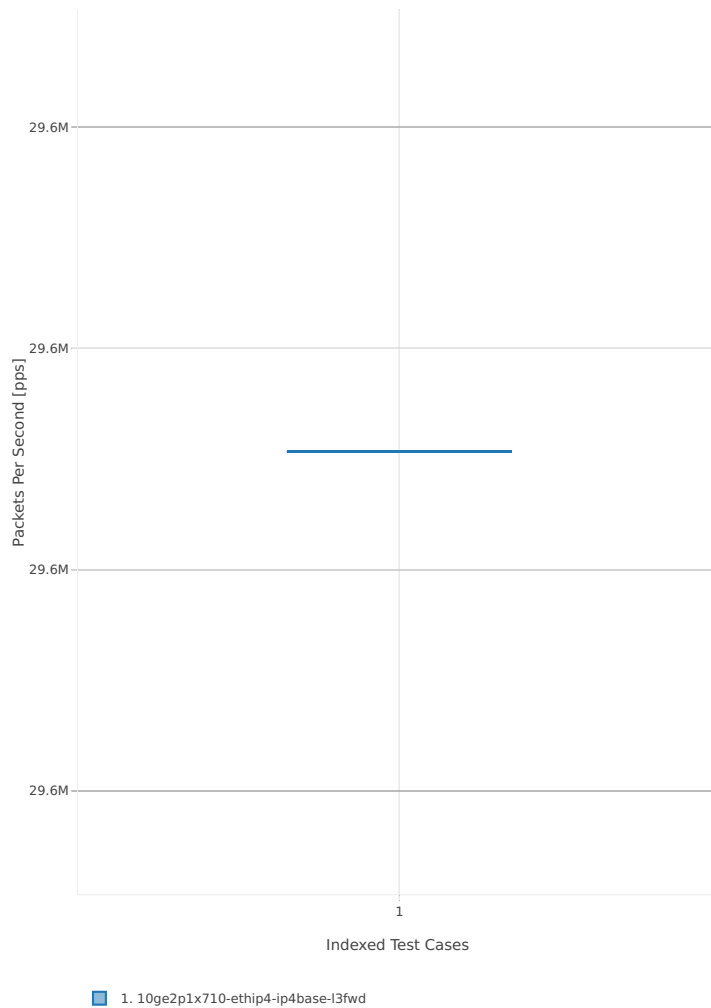
Throughput: l3fwd-3n-hsw-xl710-64b-2t2c-base-pdr



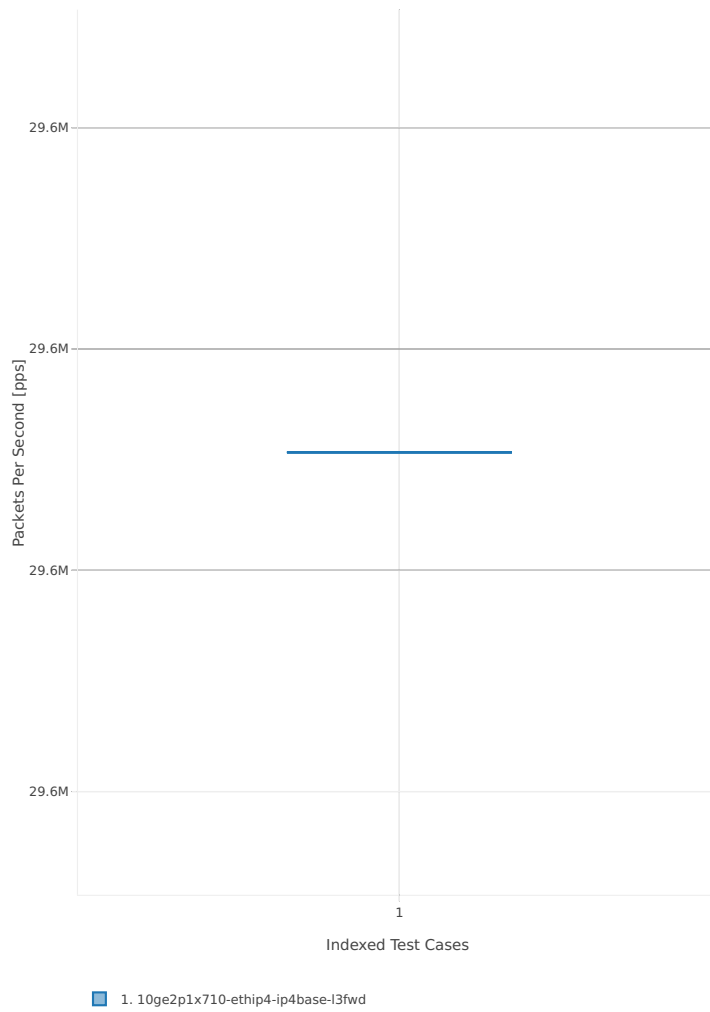
3n-skx-x710

64b-2t1c-base

Throughput: l3fwd-3n-skx-x710-64b-2t1c-base-ndr

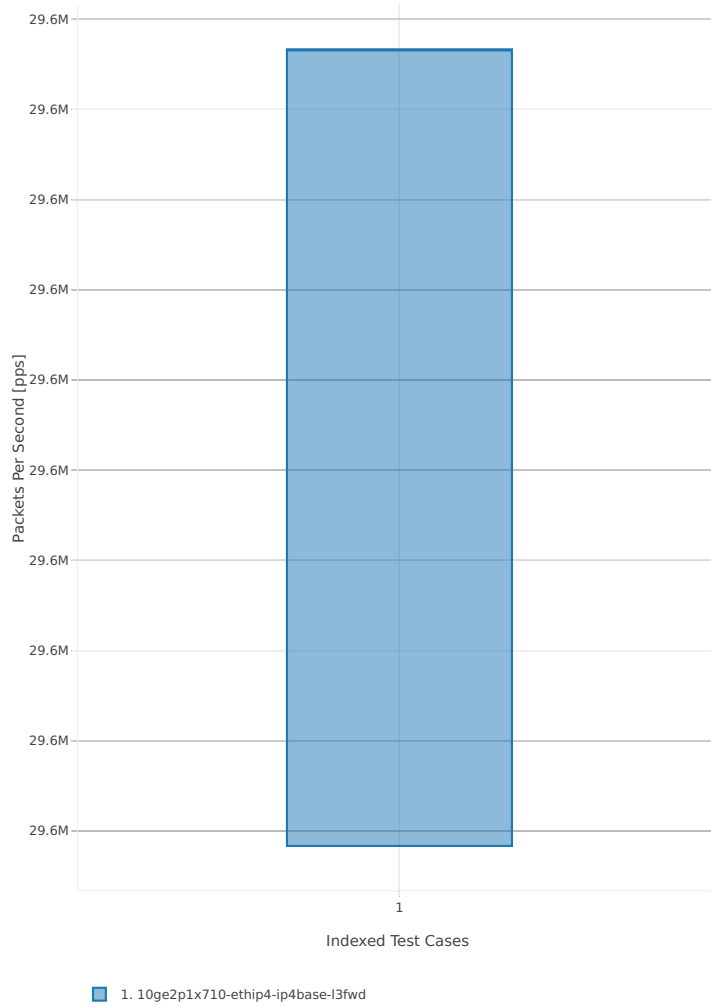


Throughput: l3fwd-3n-skx-x710-64b-2t1c-base-pdr

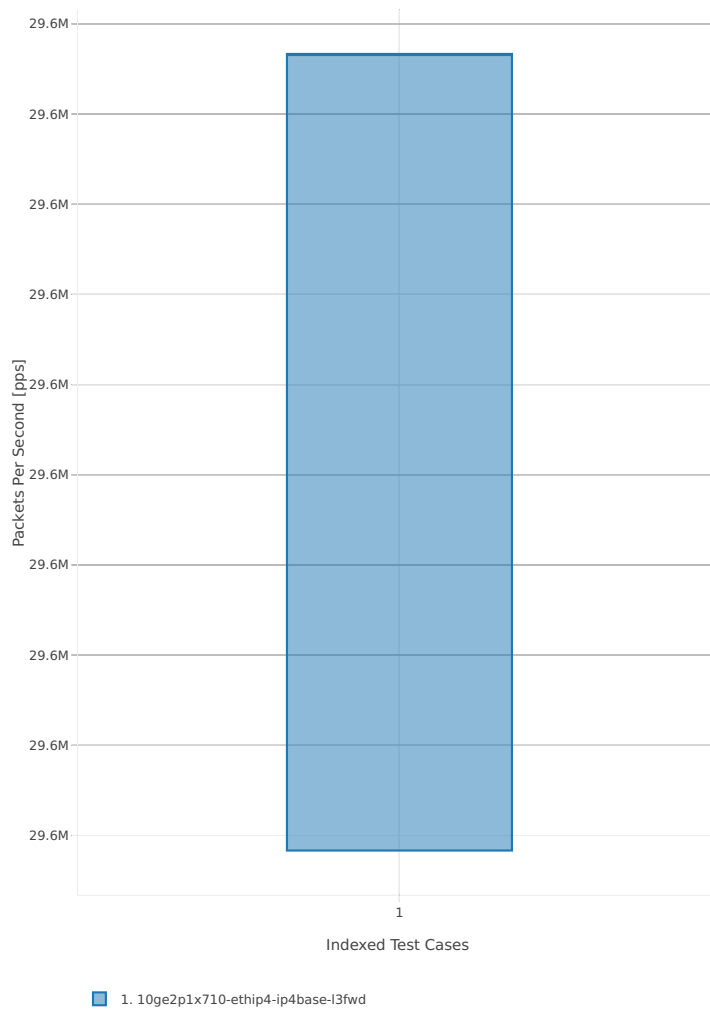


64b-4t2c-base

Throughput: l3fwd-3n-skx-x710-64b-4t2c-base-ndr



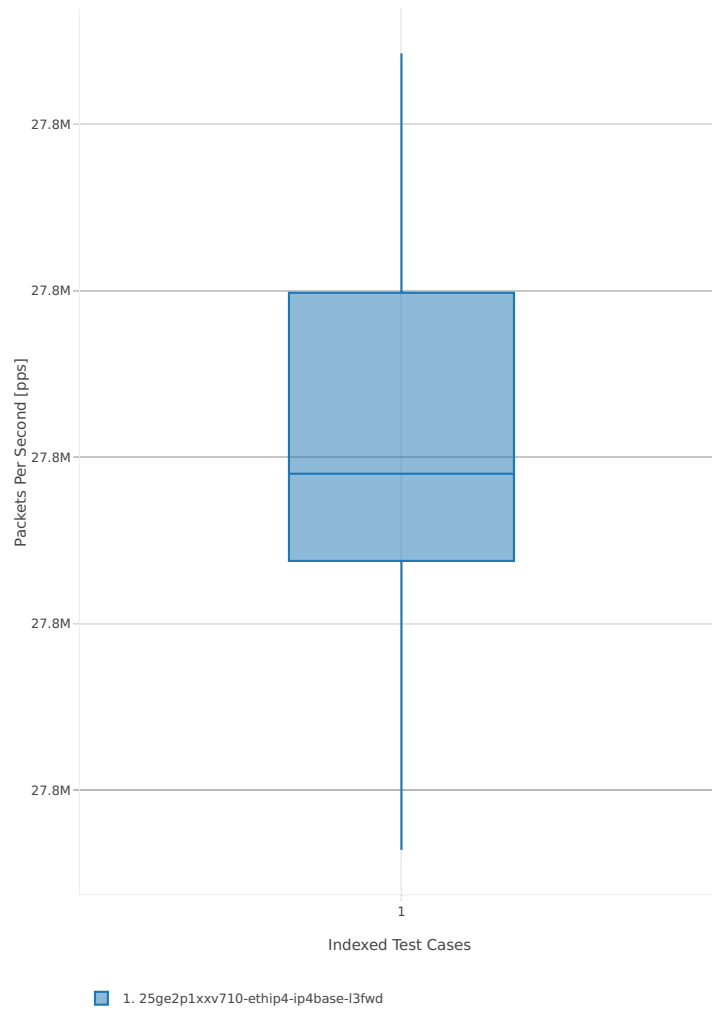
Throughput: l3fwd-3n-skx-x710-64b-4t2c-base-pdr



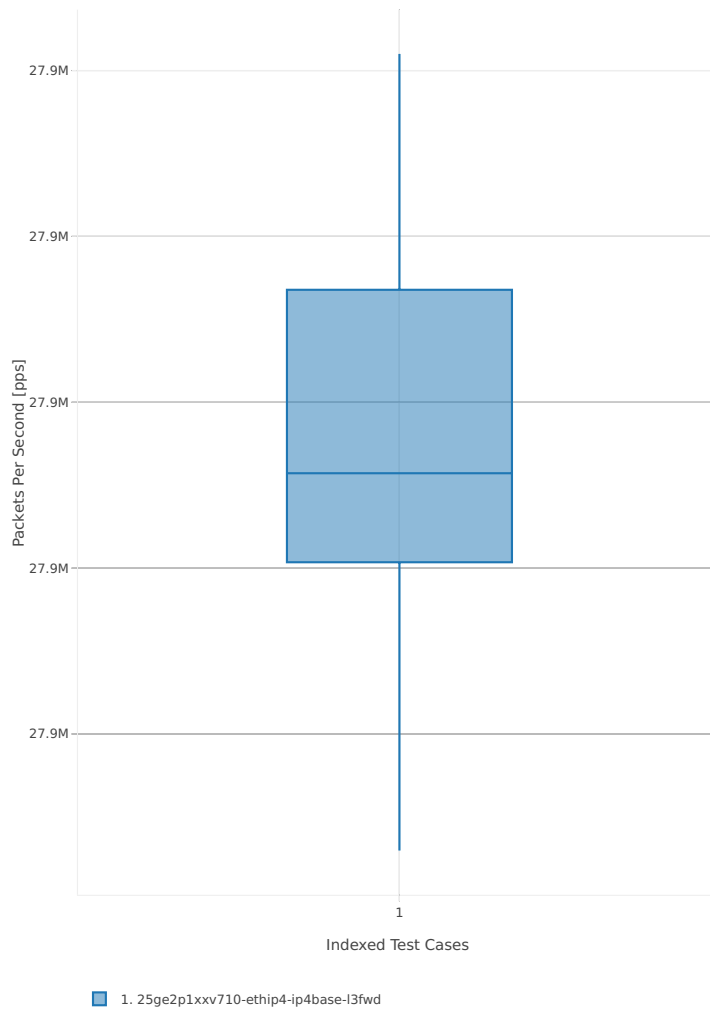
3n-skx-xxv710

64b-2t1c-base

Throughput: I3fwd-3n-skx-xxv710-64b-2t1c-base-ndr

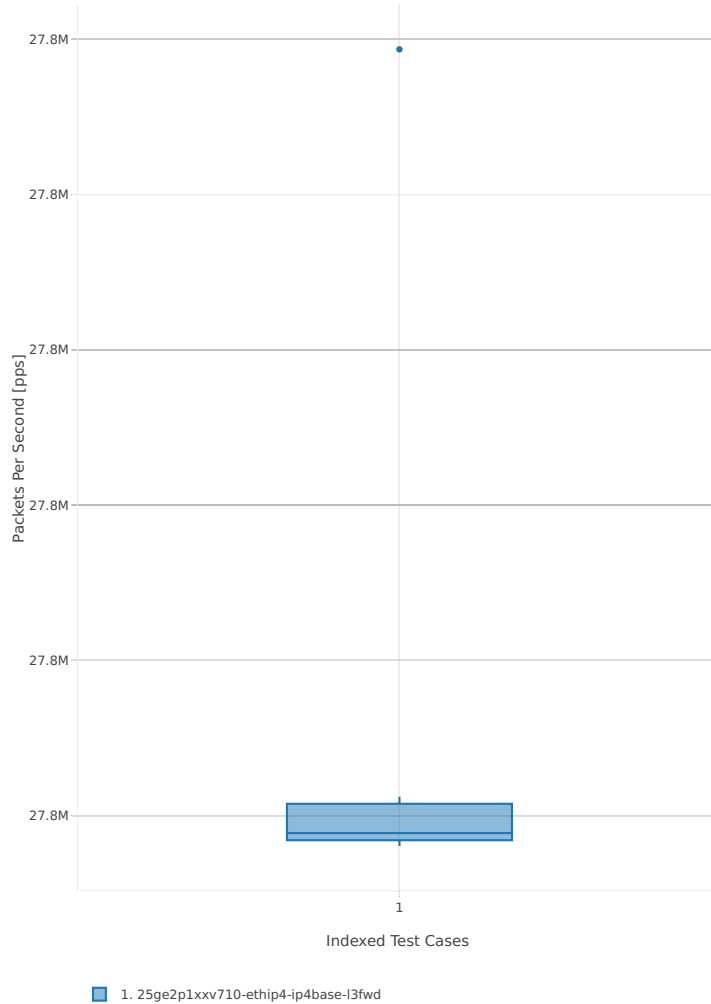


Throughput: l3fwd-3n-skx-xxv710-64b-2t1c-base-pdr

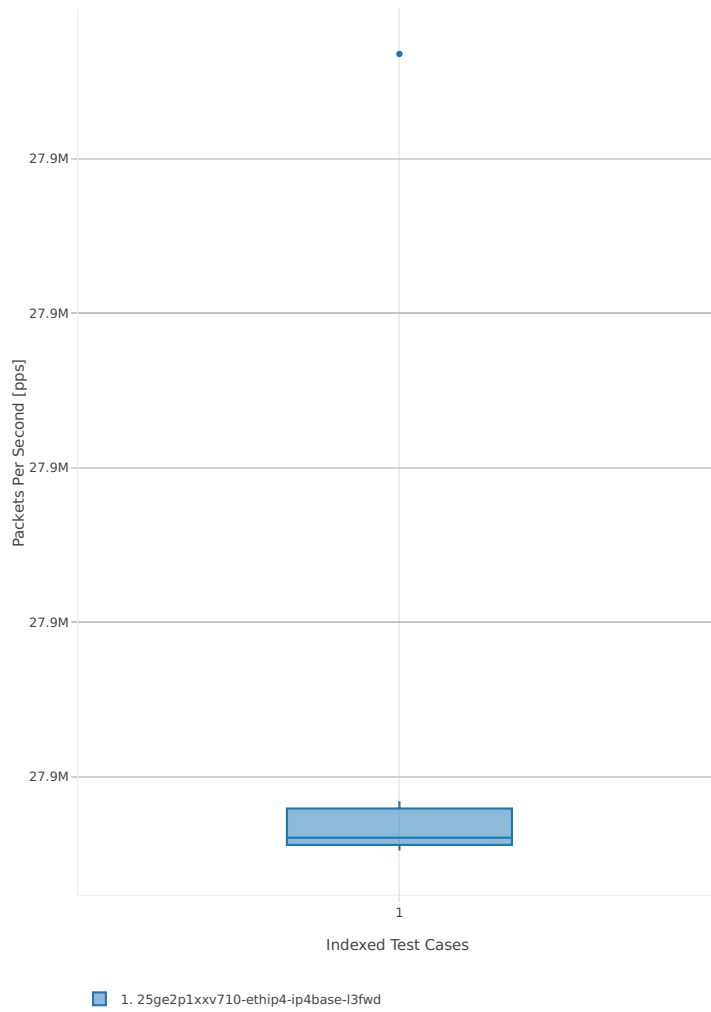


64b-4t2c-base

Throughput: I3fwd-3n-skx-xxv710-64b-4t2c-base-ndr



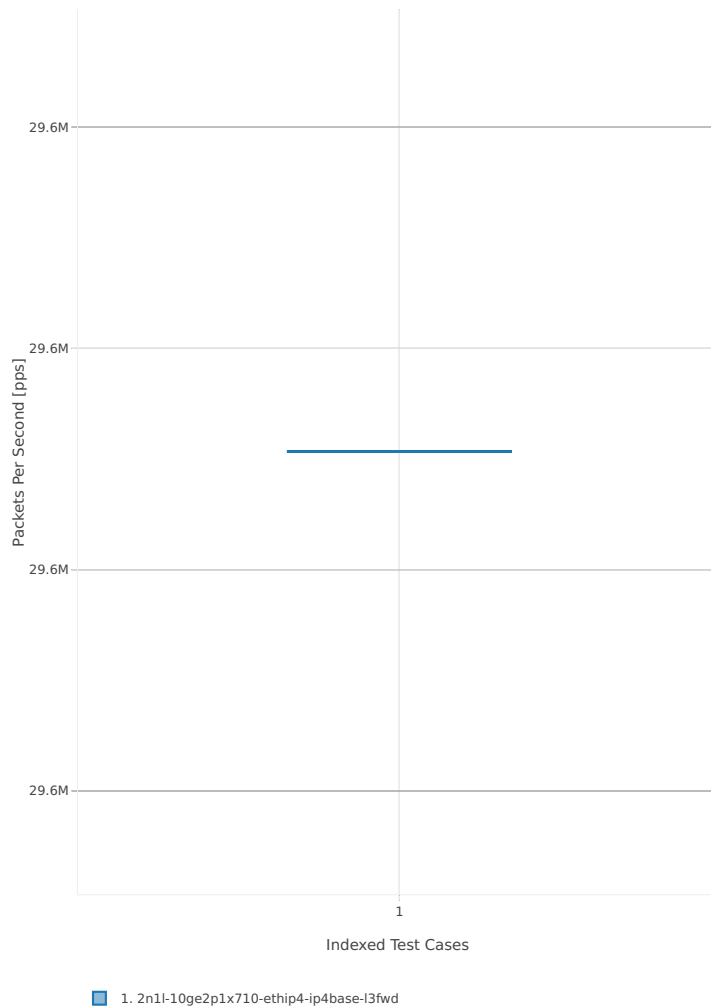
Throughput: l3fwd-3n-skx-xxv710-64b-4t2c-base-pdr



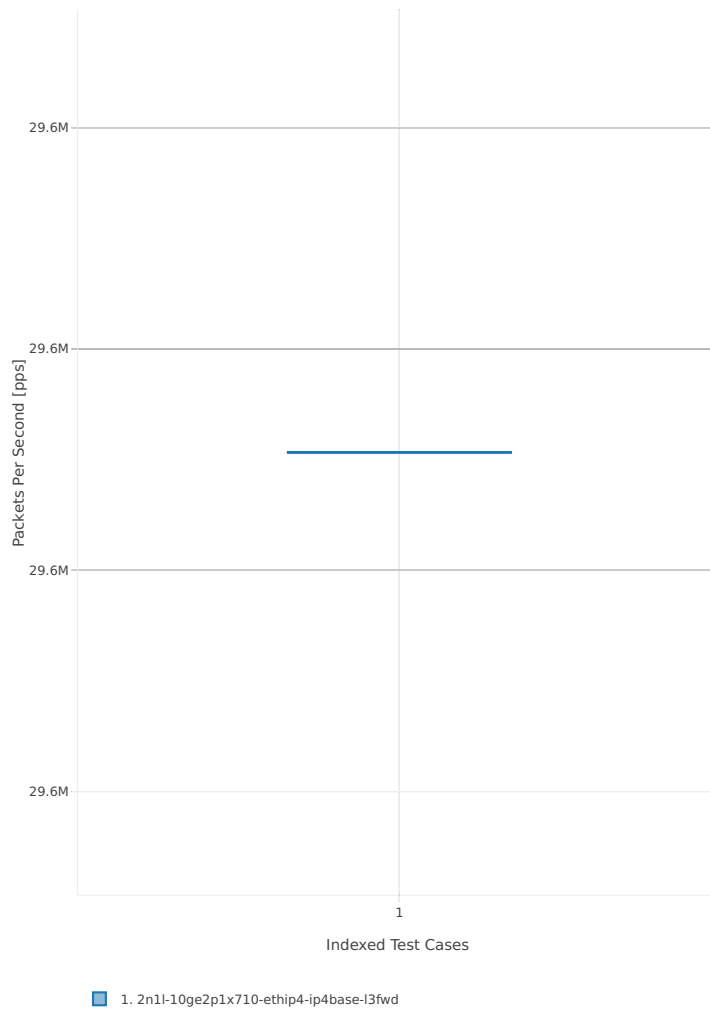
2n-skx-x710

64b-2t1c-base

Throughput: l3fwd-2n-skx-x710-64b-2t1c-base-ndr

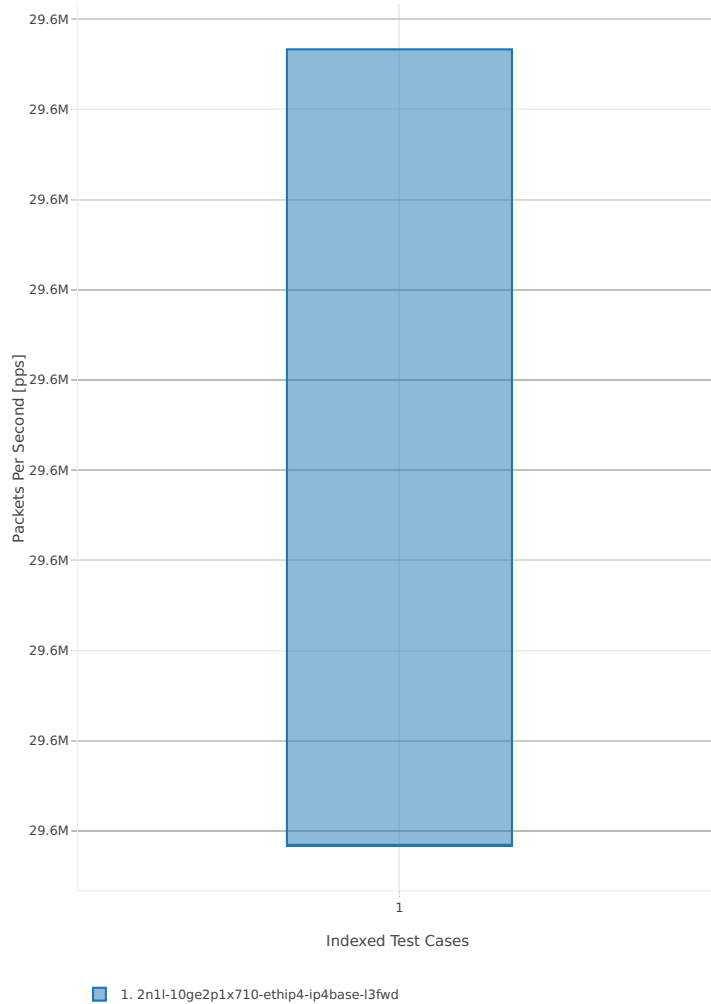


Throughput: l3fwd-2n-skx-x710-64b-2t1c-base-pdr

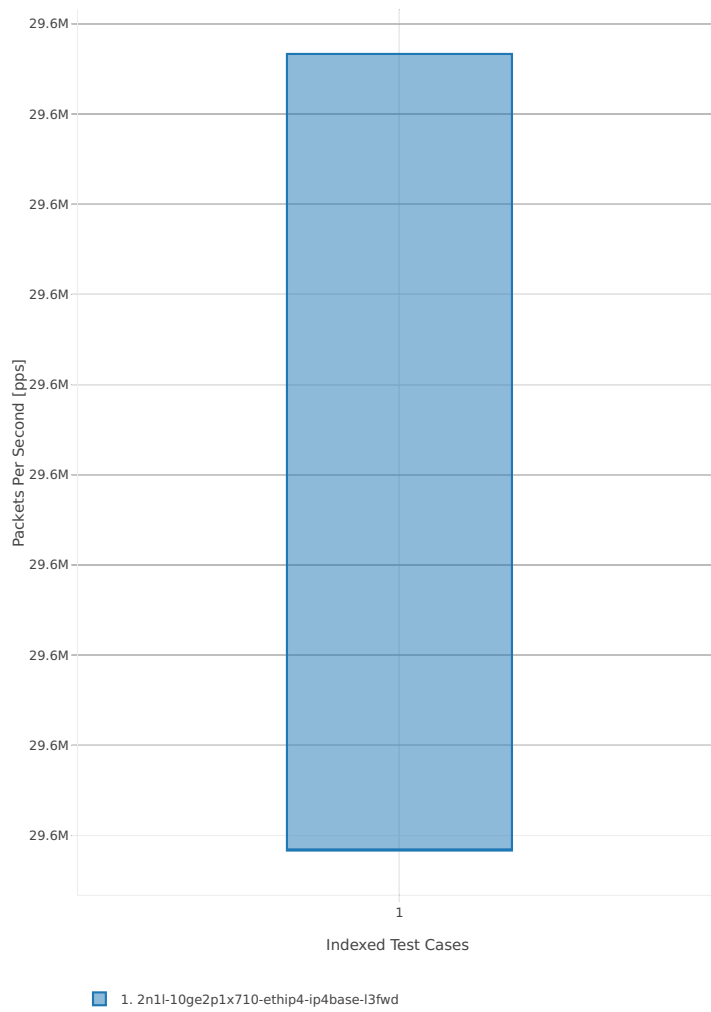


64b-4t2c-base

Throughput: l3fwd-2n-skx-x710-64b-4t2c-base-ndr



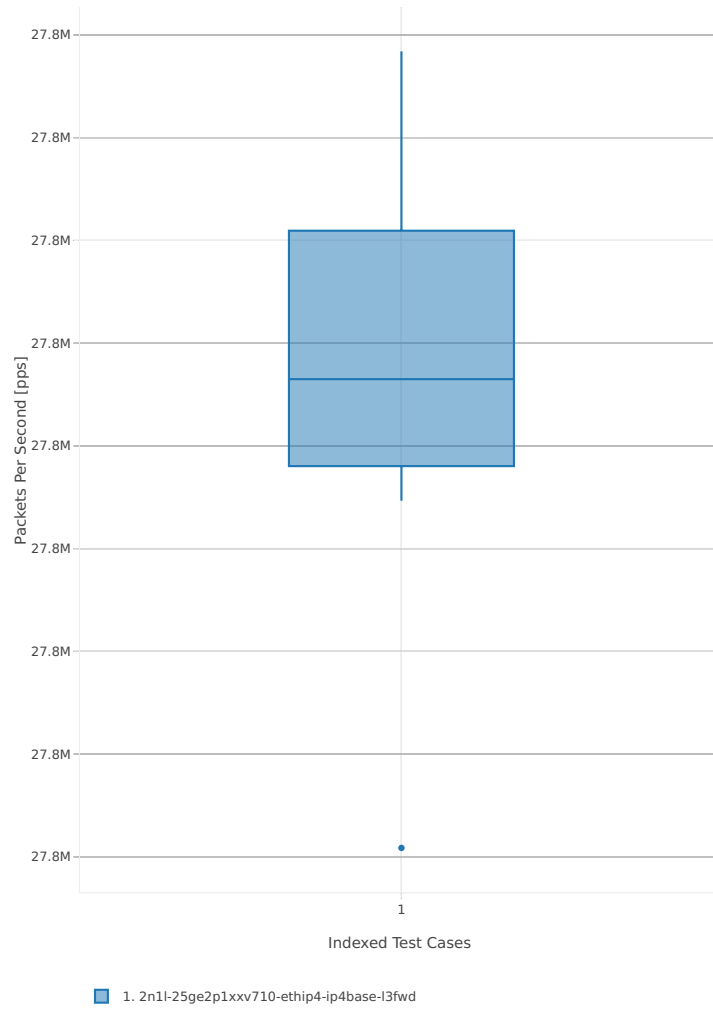
Throughput: l3fwd-2n-skx-x710-64b-4t2c-base-pdr



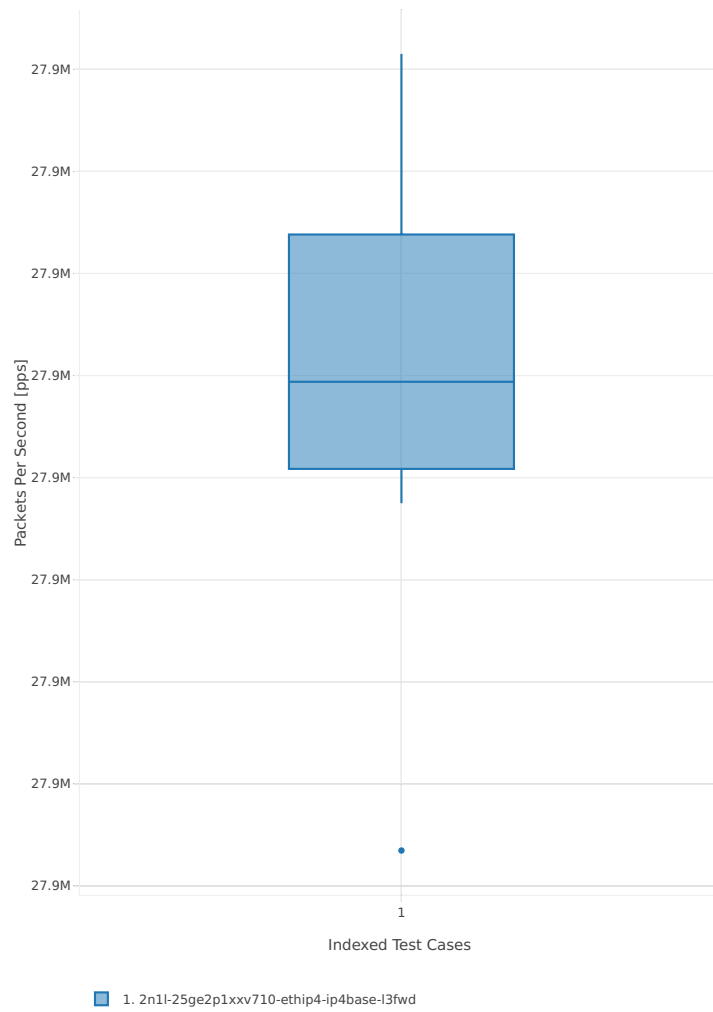
2n-skx-xxv710

64b-2t1c-base

Throughput: l3fwd-2n-skx-xxv710-64b-2t1c-base-ndr

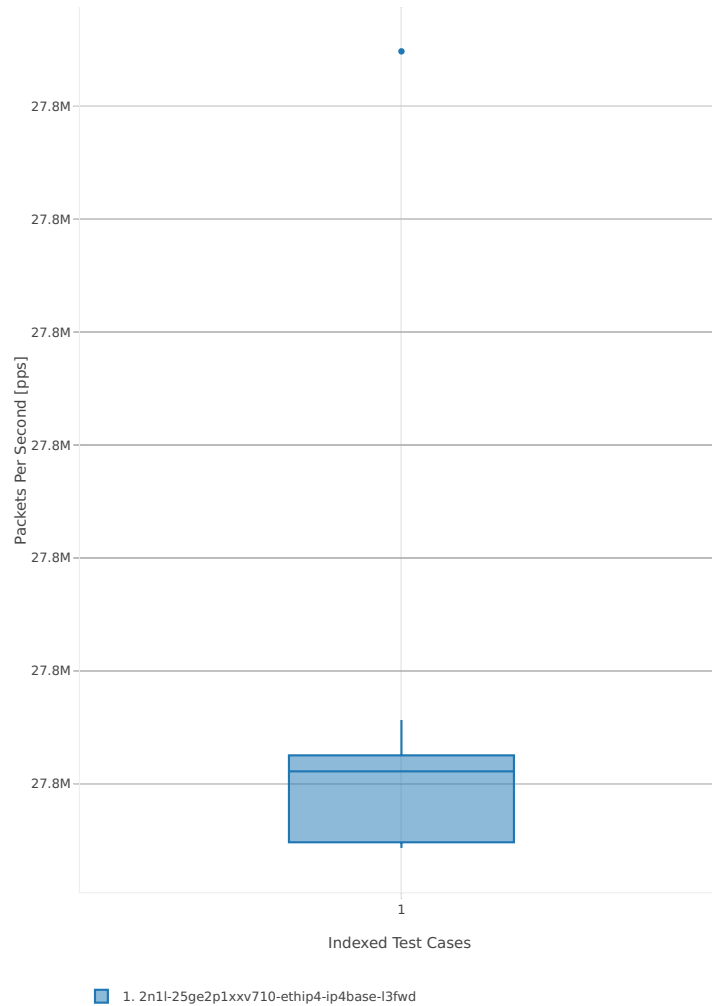


Throughput: l3fwd-2n-skx-xxv710-64b-2t1c-base-pdr

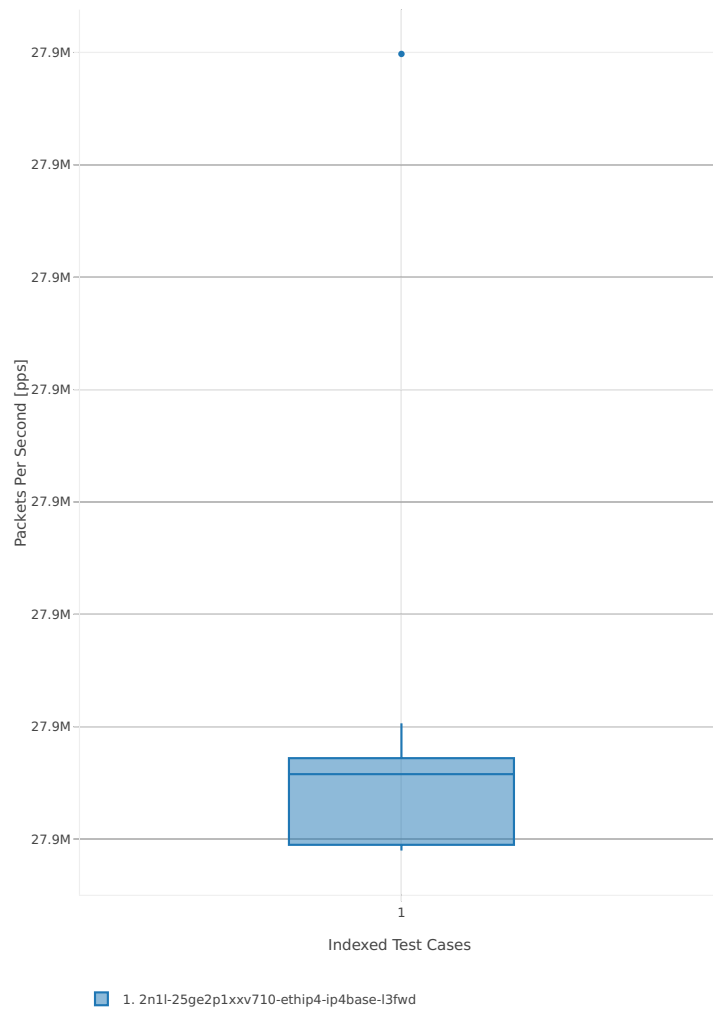


64b-4t2c-base

Throughput: l3fwd-2n-skx-xxv710-64b-4t2c-base-ndr



Throughput: l3fwd-2n-skx-xxv710-64b-4t2c-base-pdr



3.4 Packet Latency

Latency results are generated from a single execution of NDR discovery test across physical testbeds hosted LF FD.io labs: 3n-hsw, 2n-skx, 2n-skx. Box plots are used to show the Minimum, Median and Maximum packet latency per test.

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 DUT configuration.
2. **X-axis Labels:** indices of individual test suites as listed in Graph Legend and direction of latency flow:
 - West-to-East: TGint1-to-SUT1-to-SUT2-to-TGint2.
 - East-to-West: TGint2-to-SUT2-to-SUT1-to-TGint1.
3. **Y-axis Labels:** measured packet latency values in [uSec].
4. **Graph Legend:** lists X-axis indices with associated CSIT test suites executed to generate graphed test results.

Note: Test results have been generated by [FD.io test executor dpdk performance job 3n-hsw¹¹⁵](#), [FD.io test executor dpdk performance job 3n-skx¹¹⁶](#) and [FD.io test executor dpdk performance job 2n-skx¹¹⁷](#) with RF result files csit-dpdk-perf-1807-*.zip [archived here](#).

¹¹⁵ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-hsw>

¹¹⁶ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-skx>

¹¹⁷ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-2n-skx>

3.4.1 Testpmd

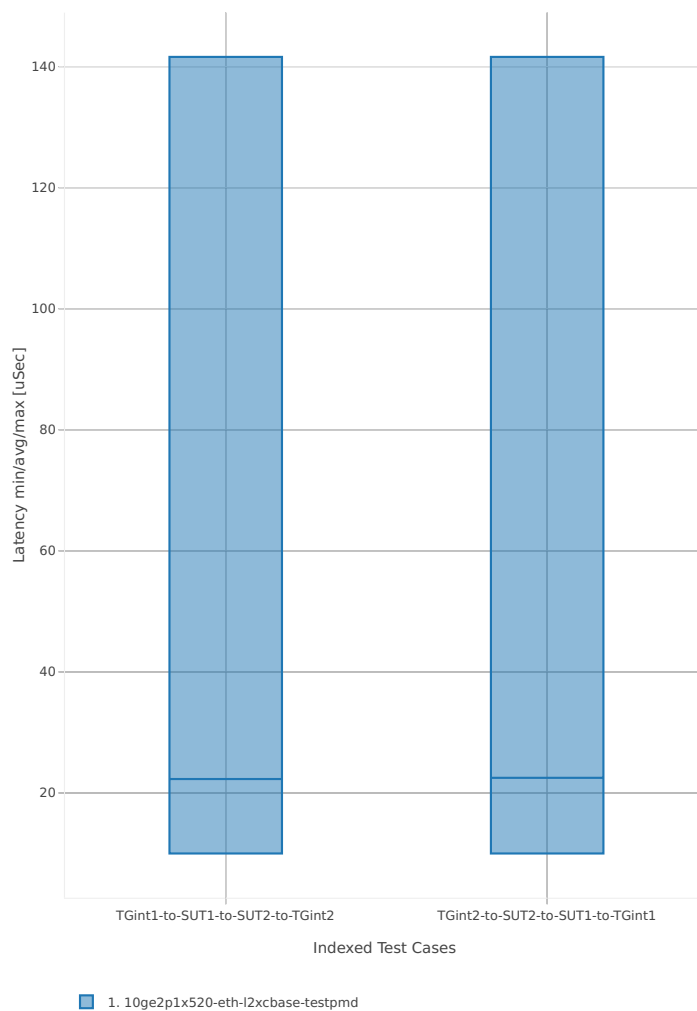
This section includes summary graphs of Testpmd Phy-to-Phy packet latency with L2 Ethernet Interface Loop measured at 100% of discovered NDR throughput rate. Latency is reported for Testpmd running in multiple configurations of Testpmd pmd thread(s), a.k.a. Testpmd data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)¹¹⁸.

3n-hsw-x520

64b-1t1c-base

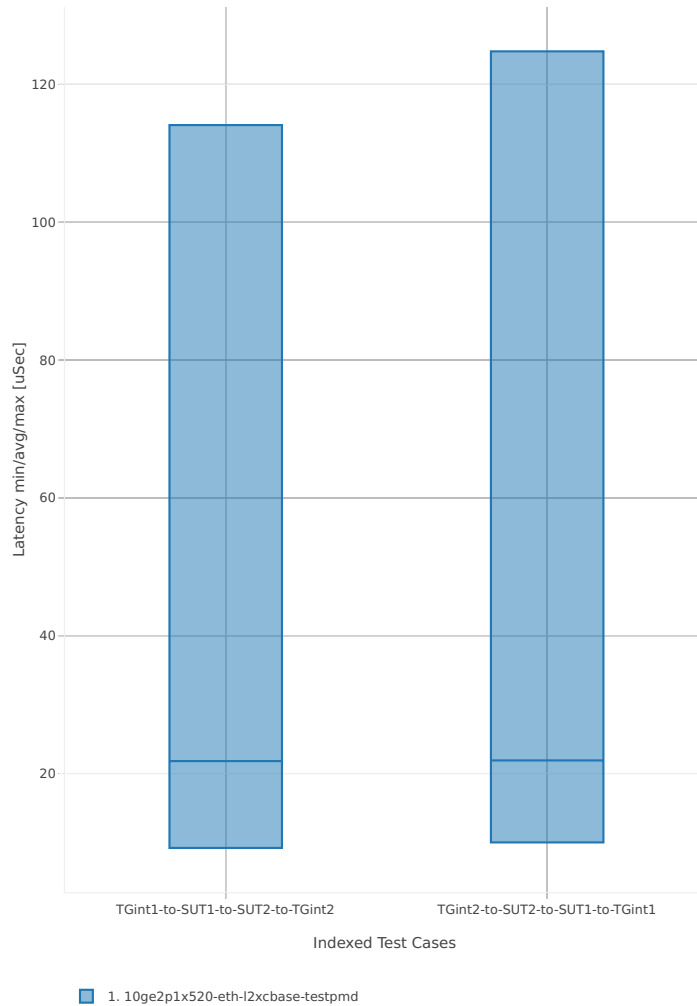
Latency: testpmd-3n-hsw-x520-64b-1t1c-base-ndr



¹¹⁸ <https://git.fd.io/csit/tree/tests/dpdk/perf?h=rls1807>

64b-2t2c-base

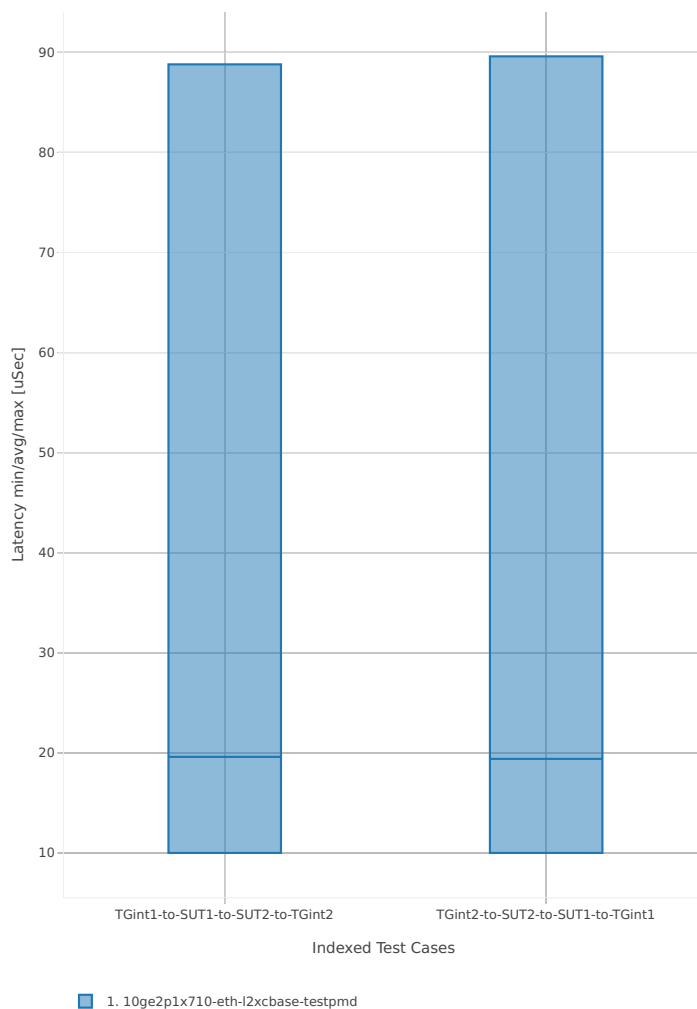
Latency: testpmd-3n-hsw-x520-64b-2t2c-base-ndr



3n-hsw-x710

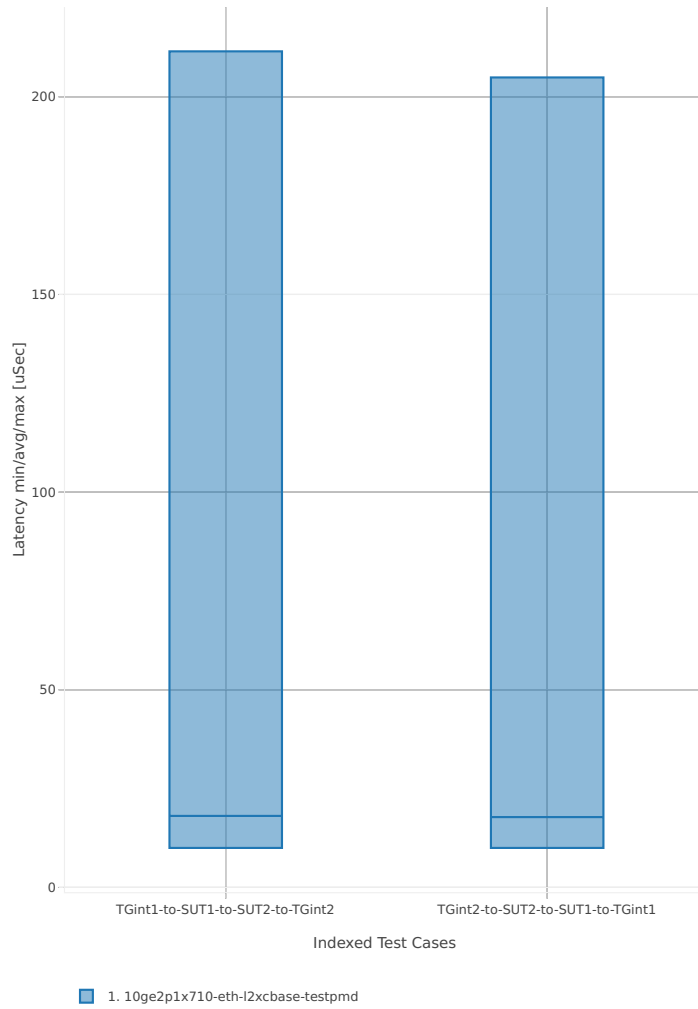
64b-1t1c-base

Latency: testpmd-3n-hsw-x710-64b-1t1c-base-ndr



64b-2t2c-base

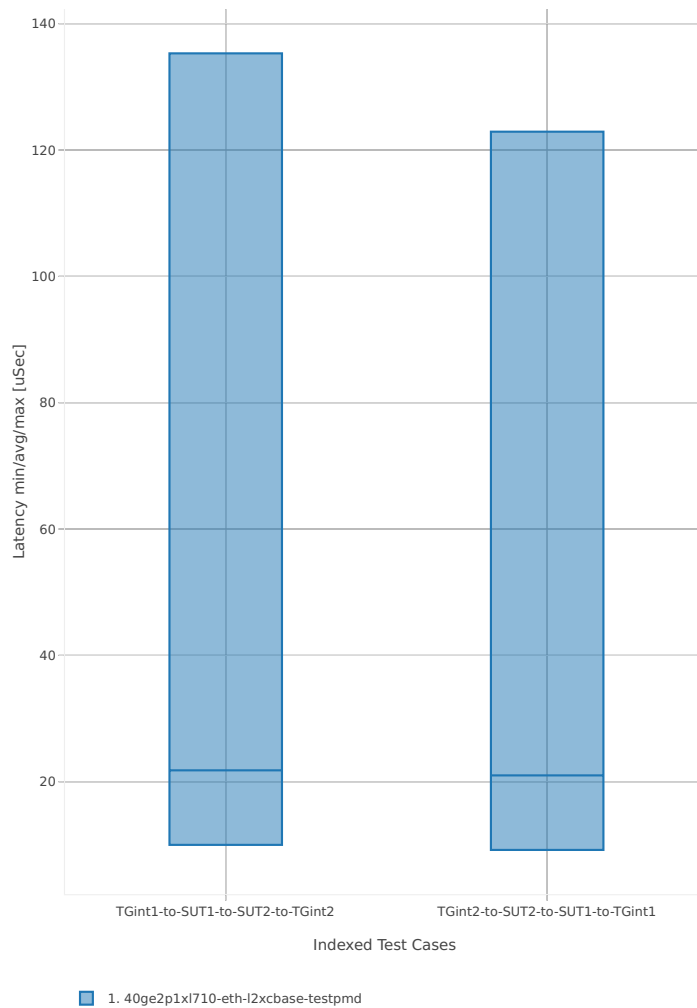
Latency: testpmd-3n-hsw-x710-64b-2t2c-base-ndr



3n-hsw-xl710

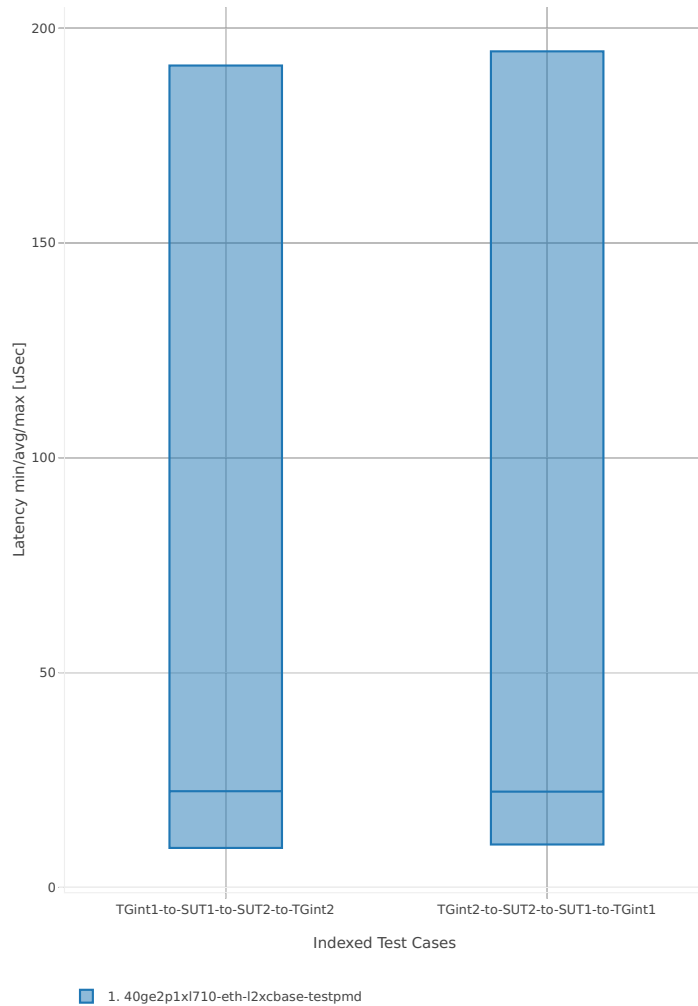
64b-1t1c-base

Latency: testpmd-3n-hsw-xl710-64b-1t1c-base-ndr



64b-2t2c-base

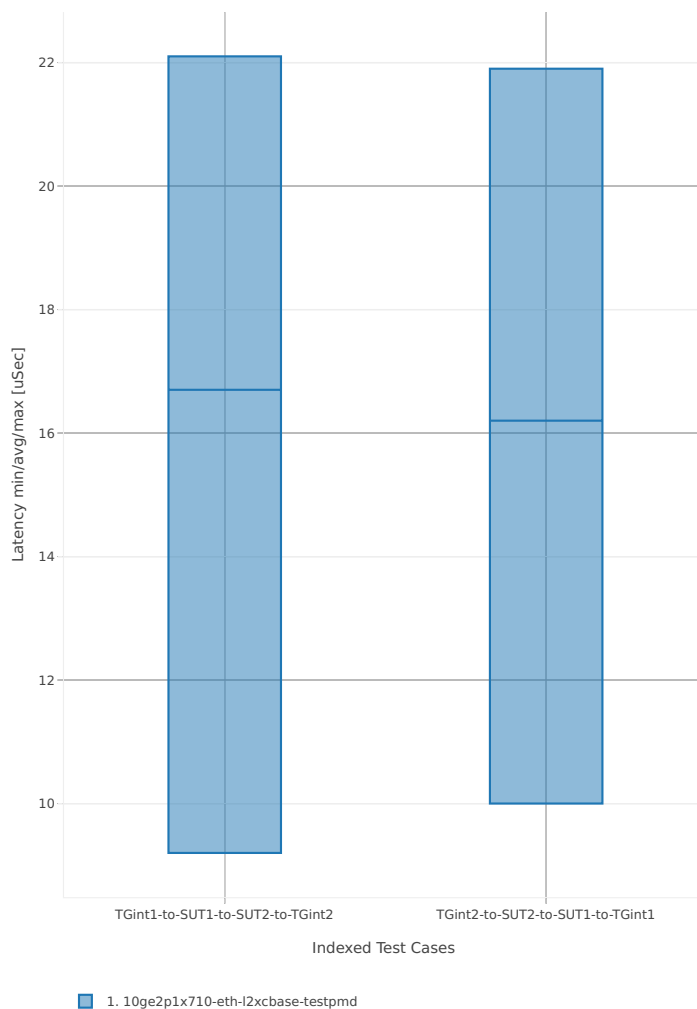
Latency: testpmd-3n-hsw-xl710-64b-2t2c-base-ndr



3n-skx-x710

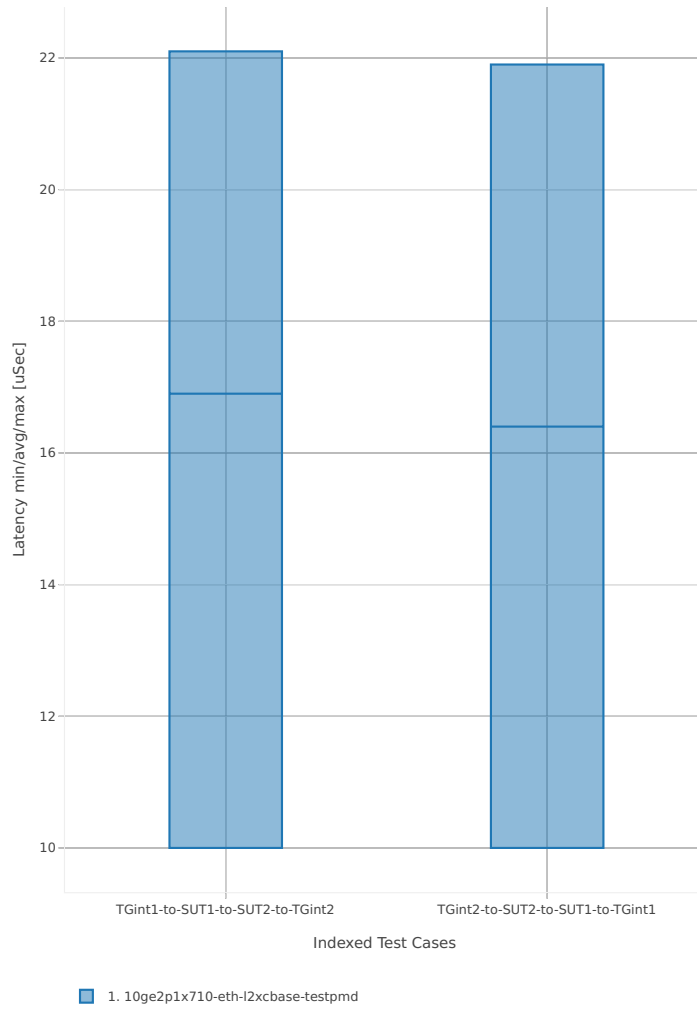
64b-2t1c-base

Latency: testpmd-3n-skx-x710-64b-2t1c-base-ndr



64b-4t2c-base

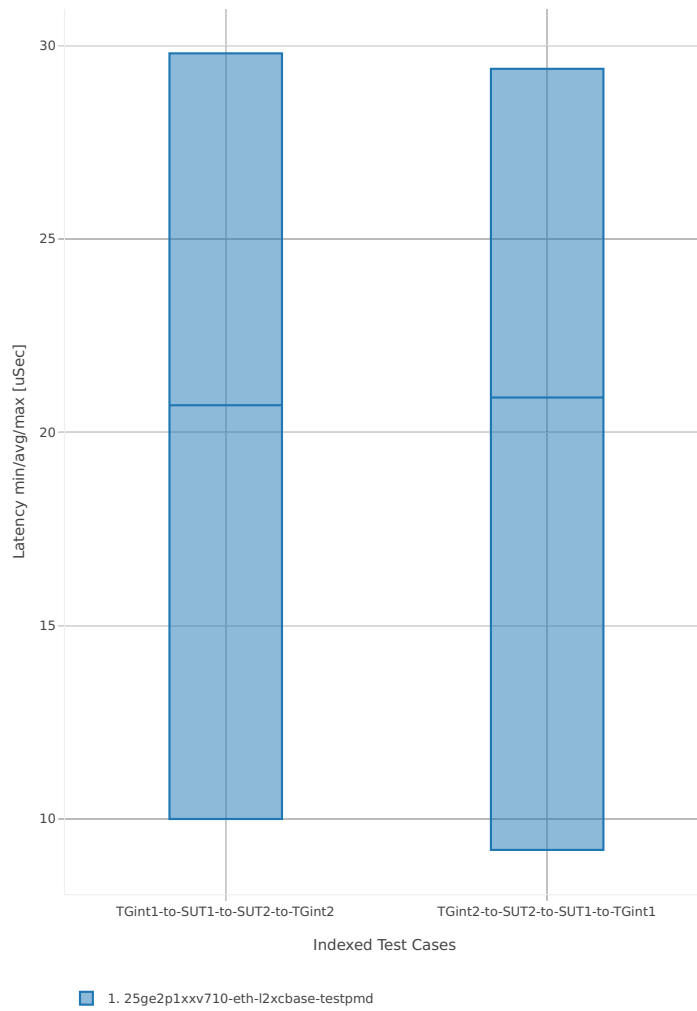
Latency: testpmd-3n-skx-x710-64b-4t2c-base-ndr



3n-skx-xxv710

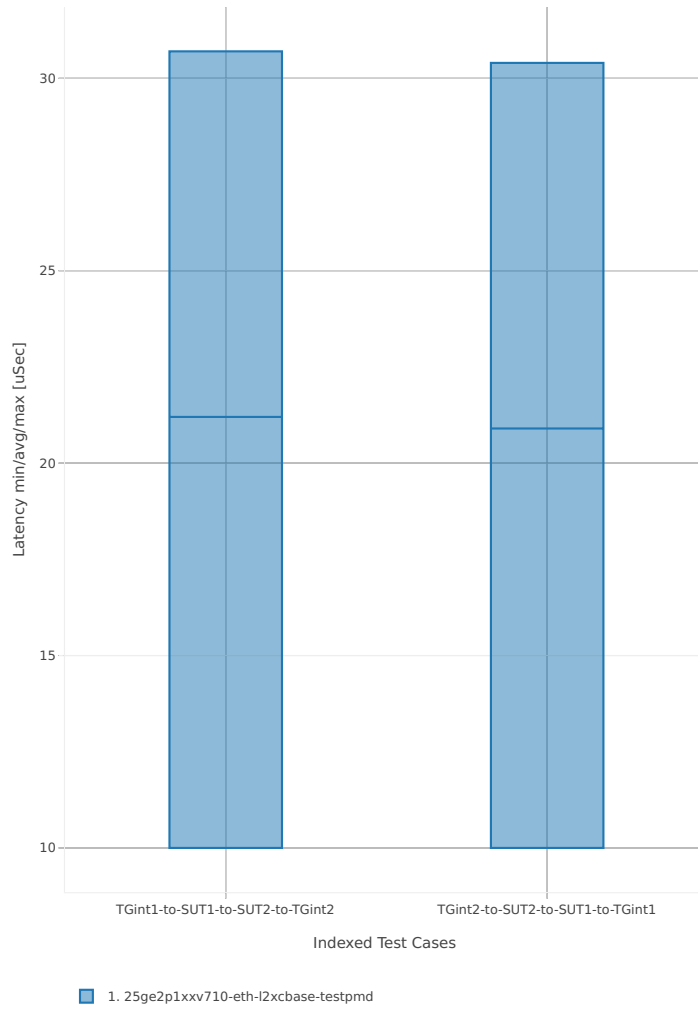
64b-2t1c-base

Latency: testpmd-3n-skx-xxv710-64b-2t1c-base-ndr



64b-4t2c-base

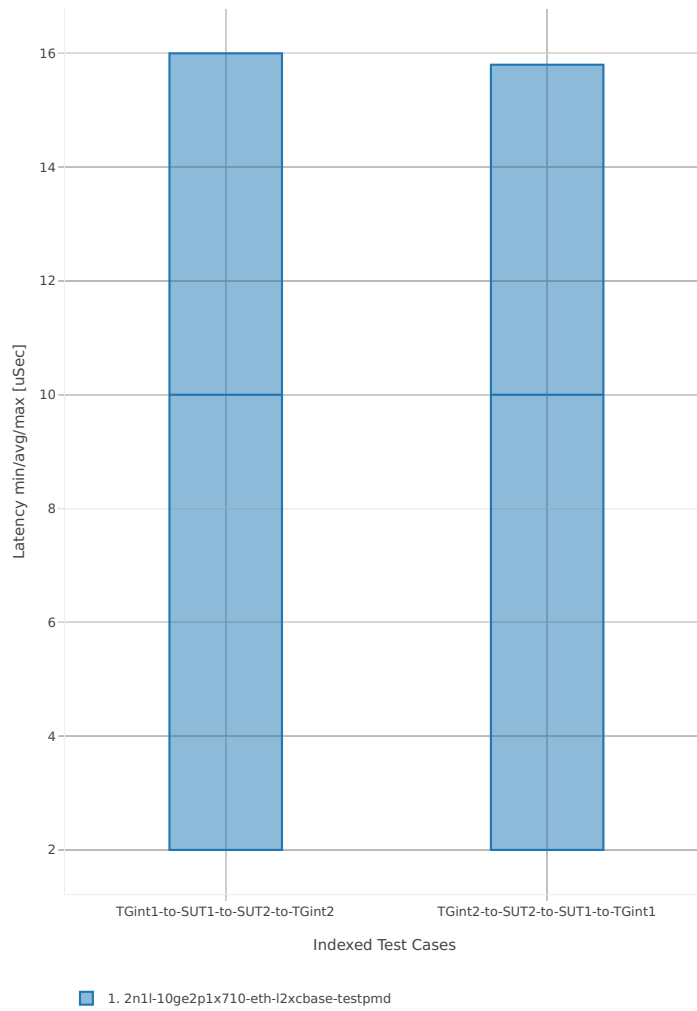
Latency: testpmd-3n-skx-xxv710-64b-4t2c-base-ndr



2n-skx-x710

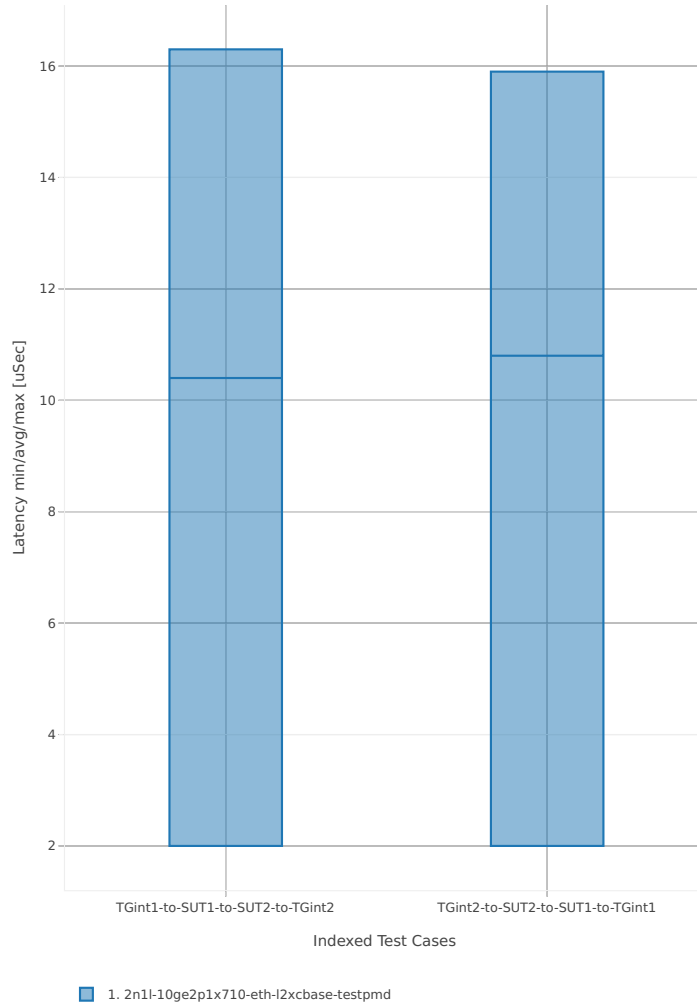
64b-2t1c-base

Latency: testpmd-2n-skx-x710-64b-2t1c-base-ndr



64b-4t2c-base

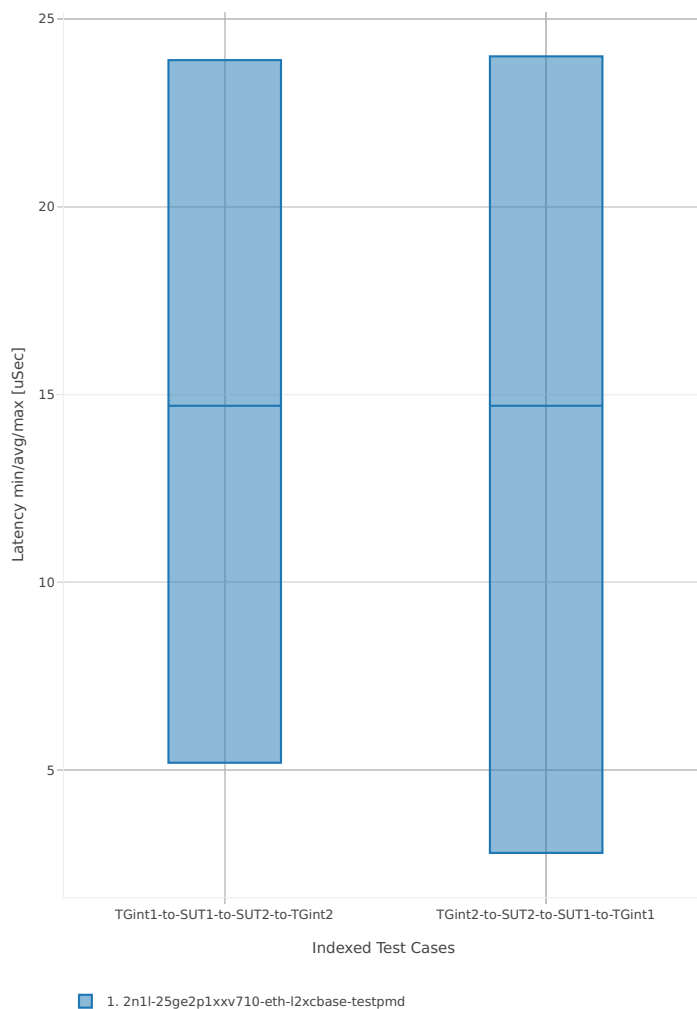
Latency: testpmd-2n-skx-x710-64b-4t2c-base-ndr



2n-skx-xxv710

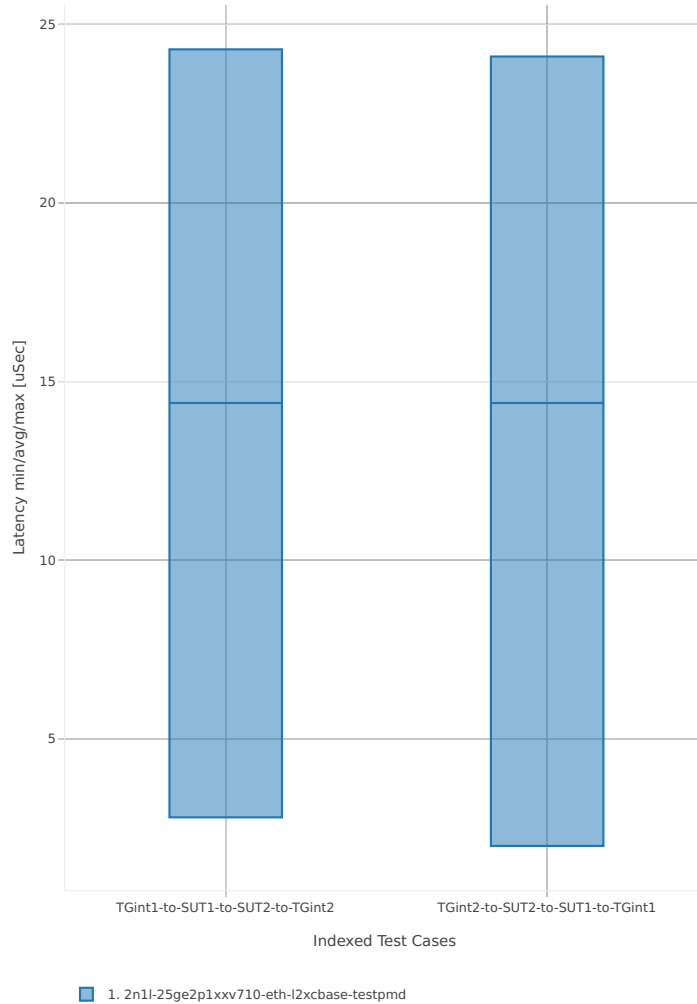
64b-2t1c-base

Latency: testpmd-2n-skx-xxv710-64b-2t1c-base-ndr



64b-4t2c-base

Latency: testpmd-2n-skx-xxv710-64b-4t2c-base-ndr



3.4.2 L3fwd

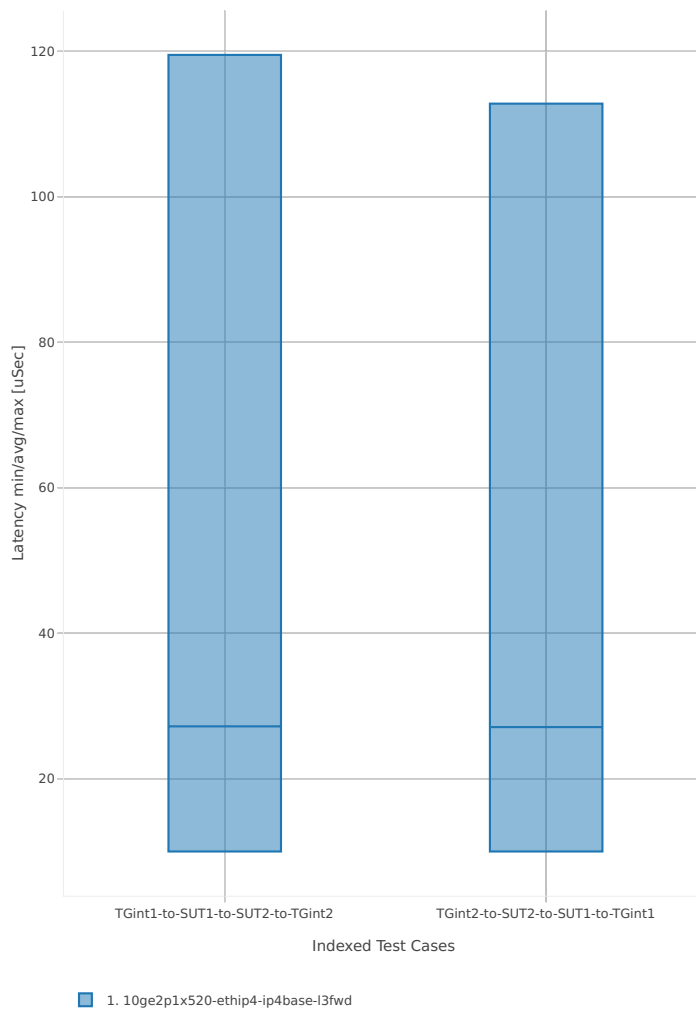
This section includes summary graphs of L3FWD Phy-to-Phy performance with packet routed forwarding measured at 100% of discovered NDR throughput rate. Latency is reported for L3FWD running in multiple configurations of L3FWD pmd thread(s), a.k.a. L3FWD data plane thread(s), and their physical CPU core(s) placement.

CSIT source code for the test cases used for plots can be found in [CSIT git repository](#)¹¹⁹.

3n-hsw-x520

64b-1t1c-base

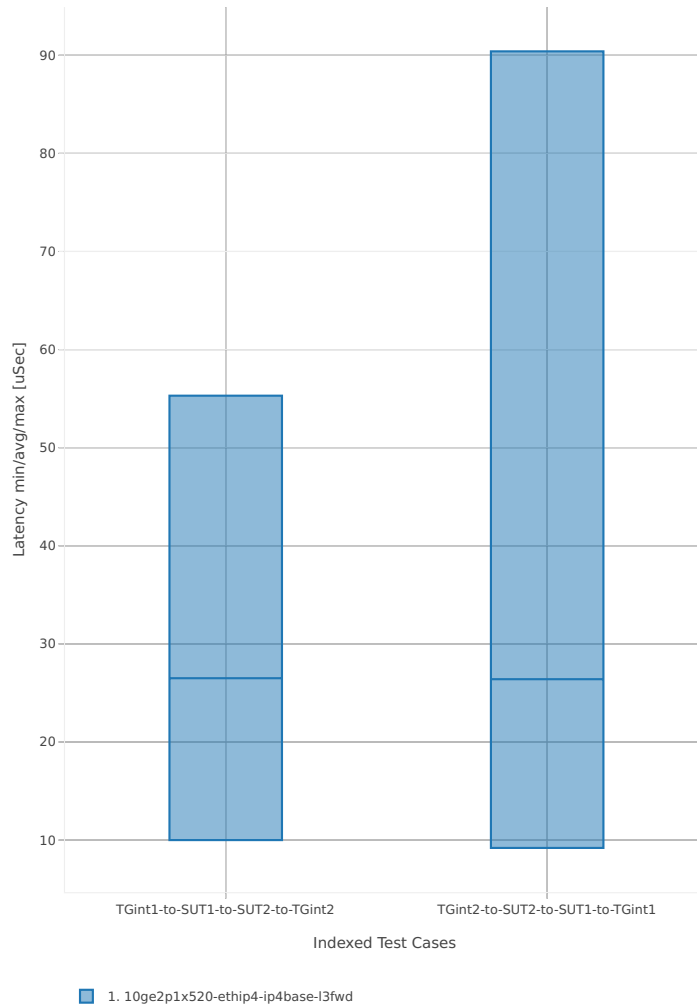
Latency: l3fwd-3n-hsw-x520-64b-1t1c-base-ndr



¹¹⁹ <https://git.fd.io/csit/tree/tests/dpdk/perf?h=rls1807>

64b-2t2c-base

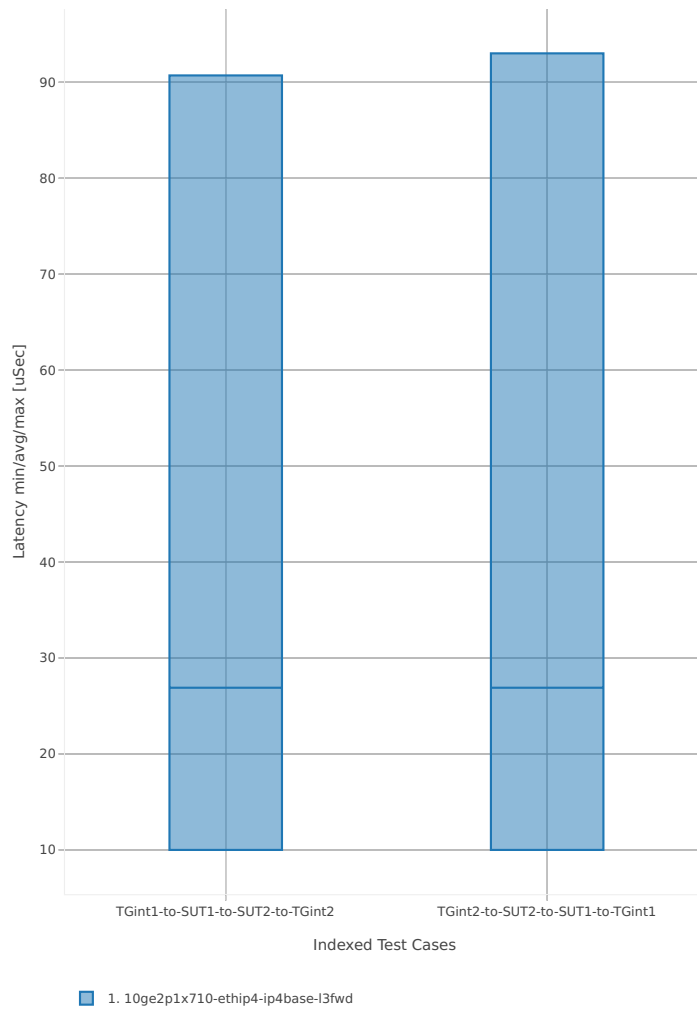
Latency: l3fwd-3n-hsw-x520-64b-2t2c-base-ndr



3n-hsw-x710

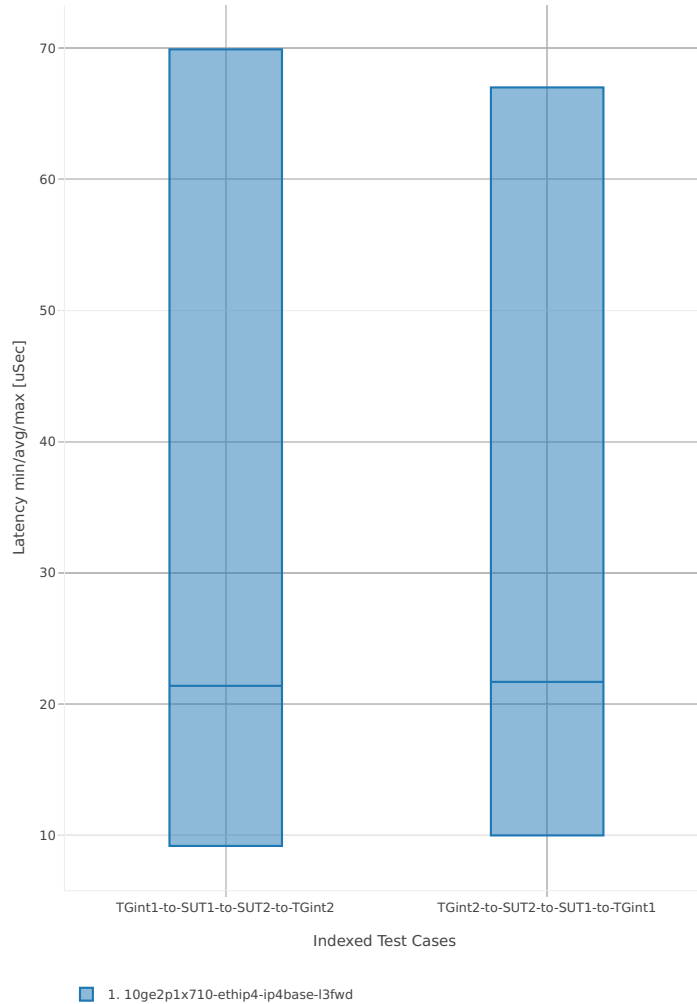
64b-1t1c-base

Latency: l3fwd-3n-hsw-x710-64b-1t1c-base-ndr



64b-2t2c-base

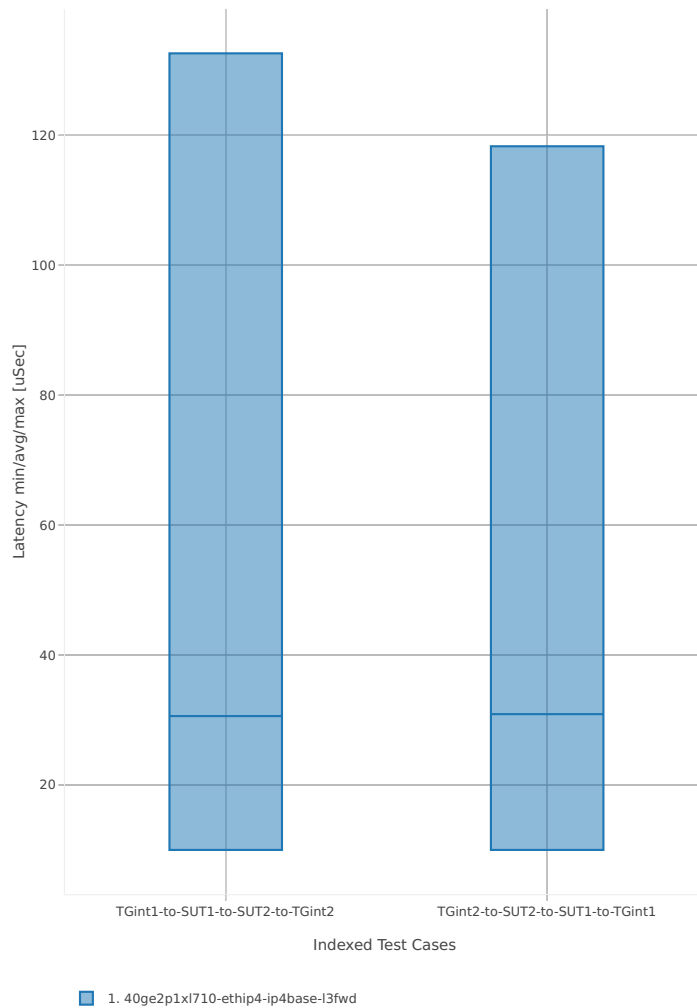
Latency: l3fwd-3n-hsw-x710-64b-2t2c-base-ndr



3n-hsw-xl710

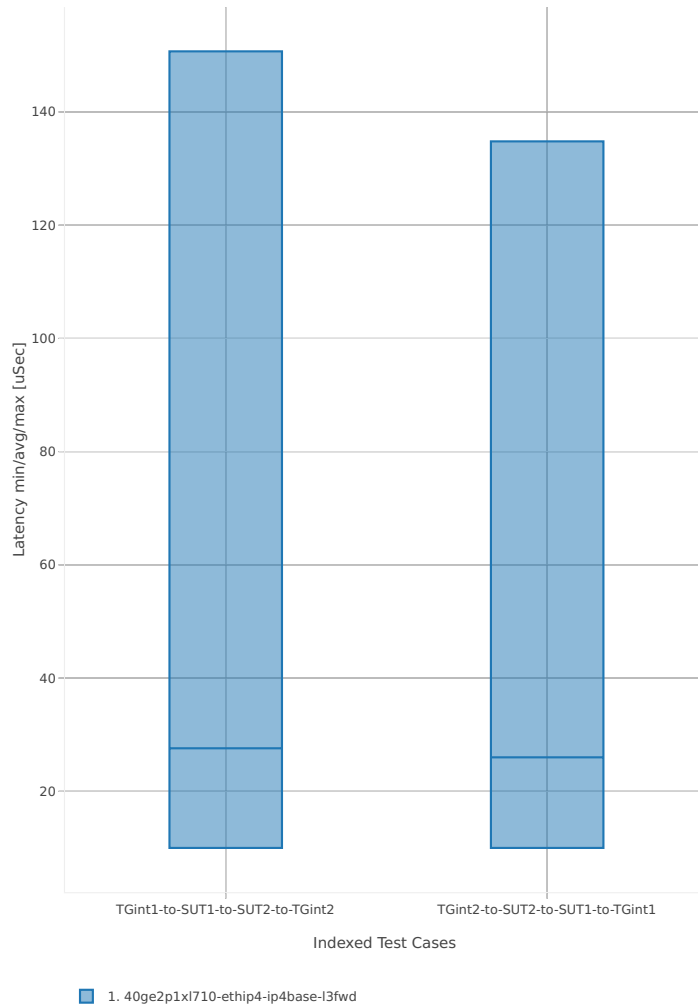
64b-1t1c-base

Latency: l3fwd-3n-hsw-xl710-64b-1t1c-base-ndr



64b-2t2c-base

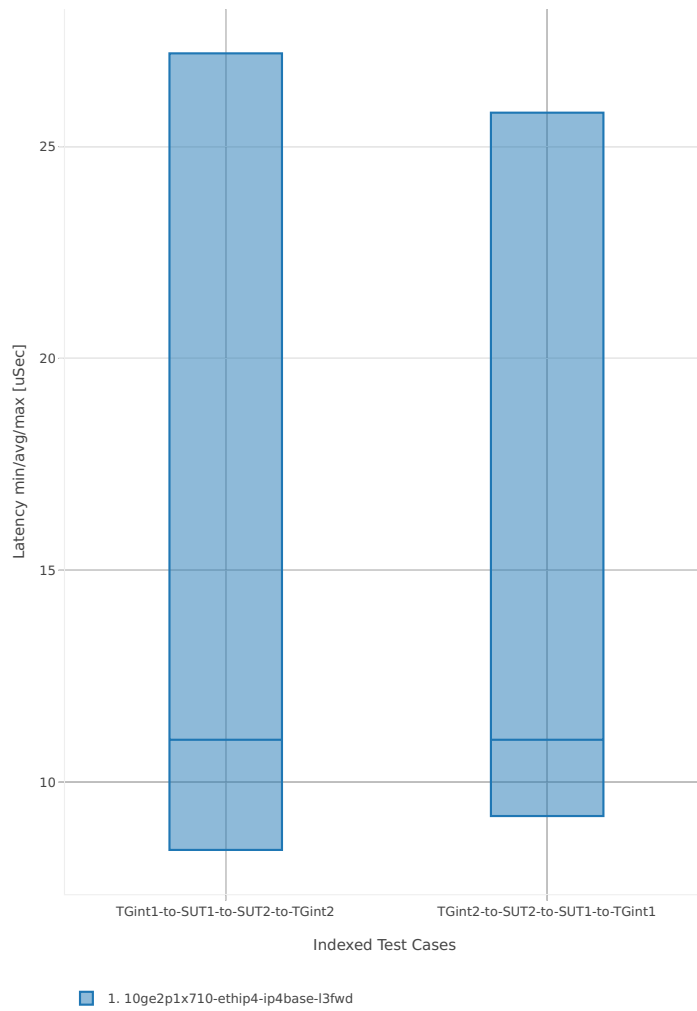
Latency: l3fwd-3n-hsw-xl710-64b-2t2c-base-ndr



3n-skx-x710

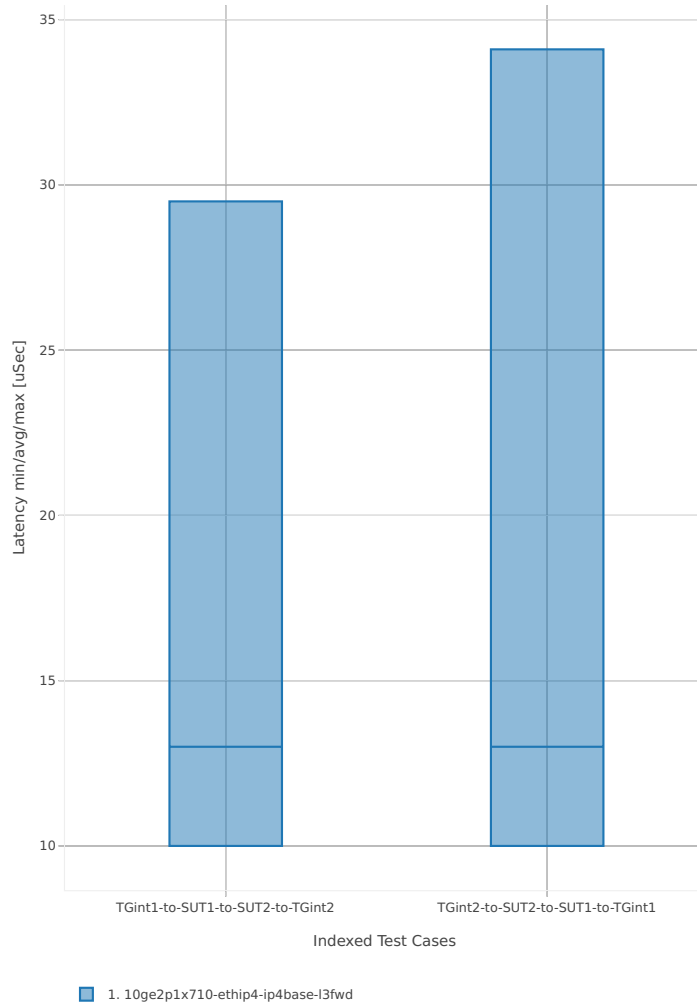
64b-2t1c-base

Latency: l3fwd-3n-skx-x710-64b-2t1c-base-ndr



64b-4t2c-base

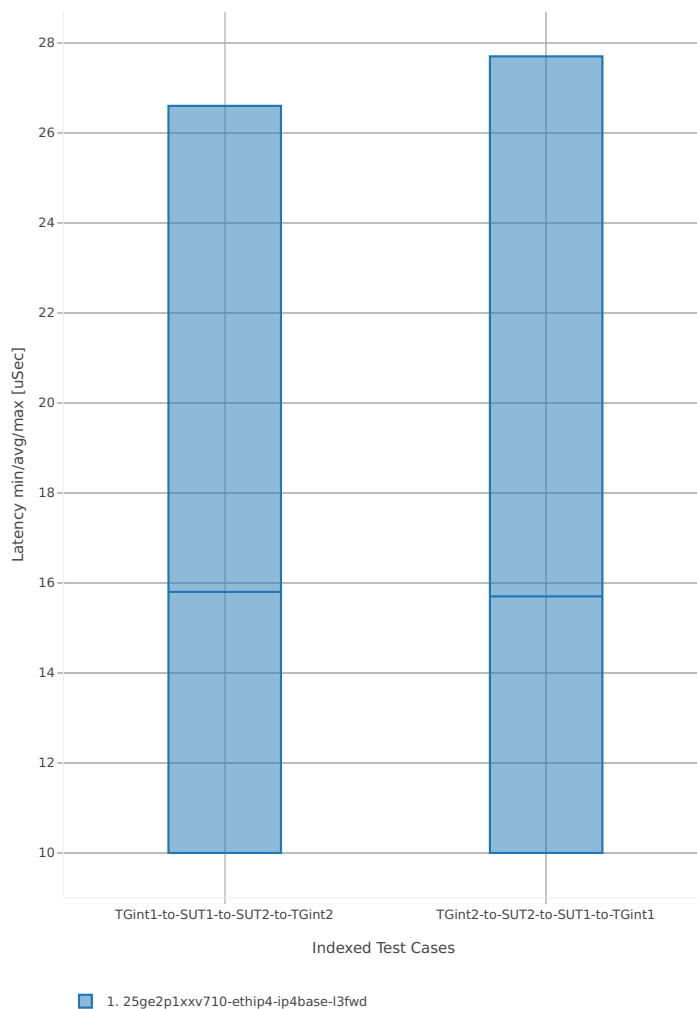
Latency: l3fwd-3n-skx-x710-64b-4t2c-base-ndr



3n-skx-xxv710

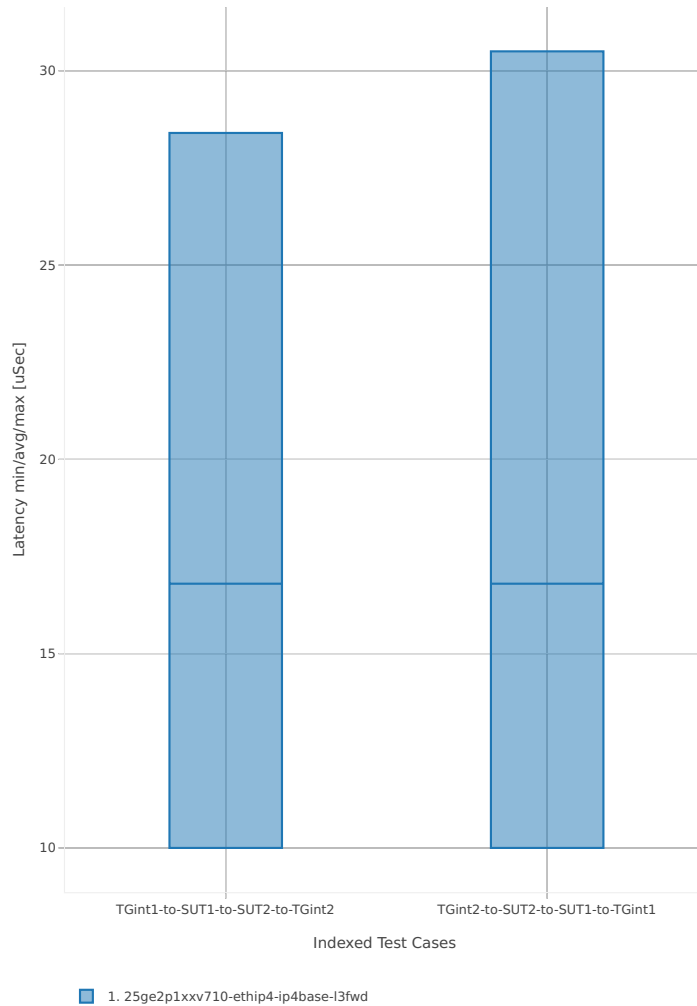
64b-2t1c-base

Latency: l3fwd-3n-skx-xxv710-64b-2t1c-base-ndr



64b-4t2c-base

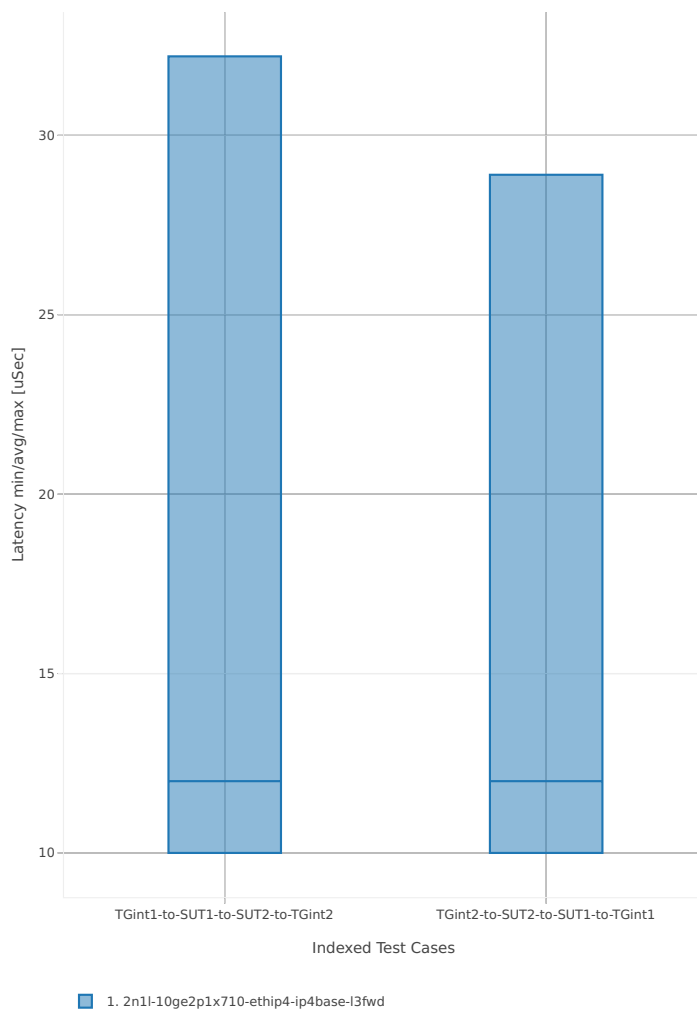
Latency: l3fwd-3n-skx-xxv710-64b-4t2c-base-ndr



2n-skx-x710

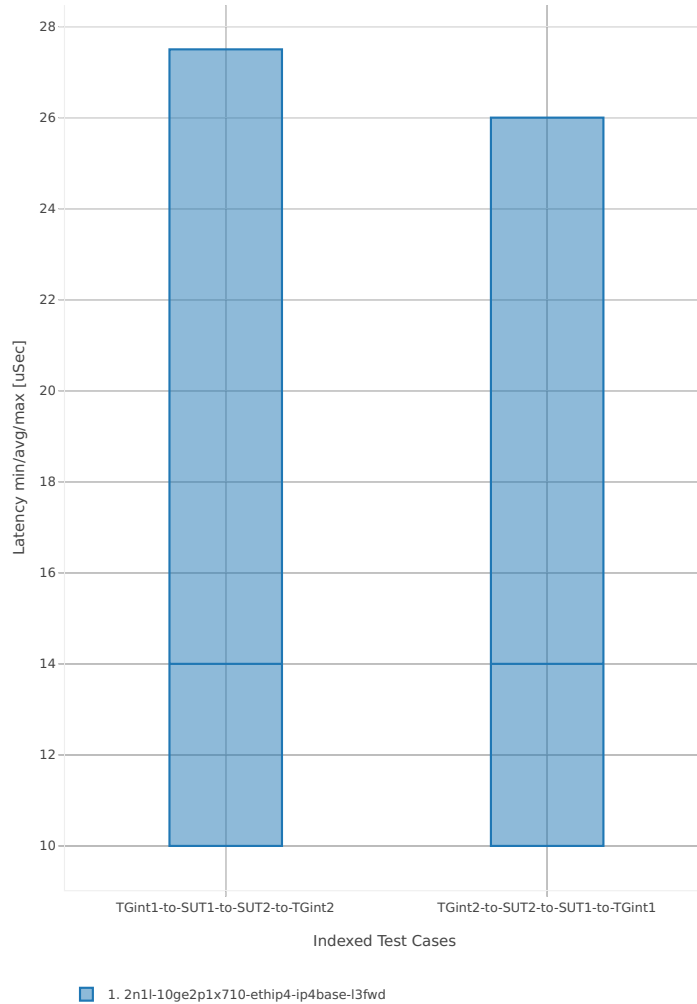
64b-2t1c-base

Latency: l3fwd-2n-skx-x710-64b-2t1c-base-ndr



64b-4t2c-base

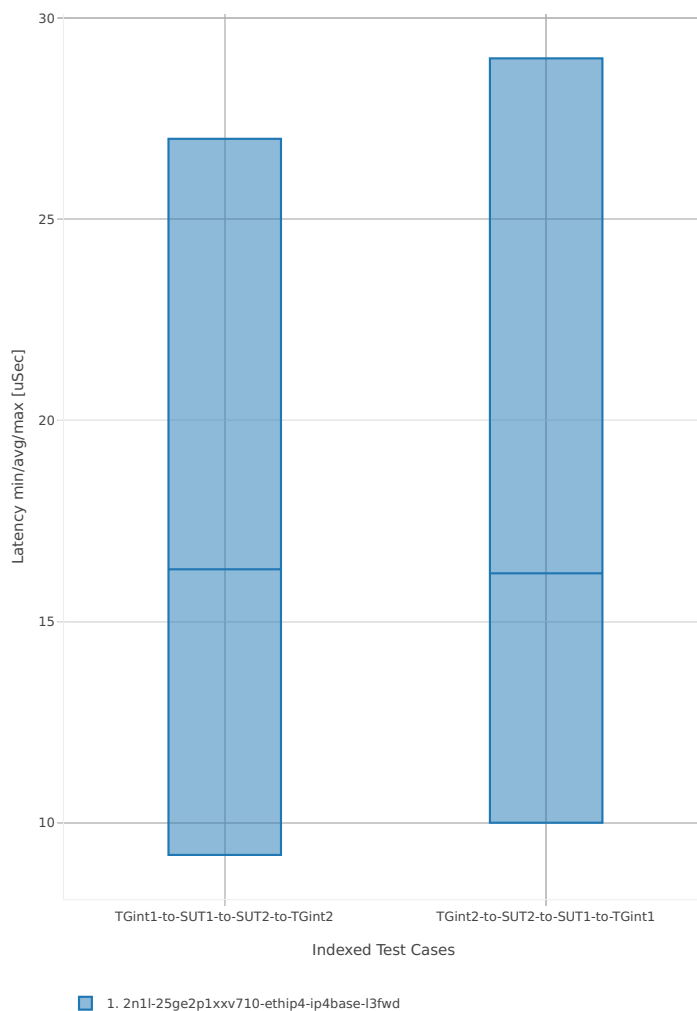
Latency: l3fwd-2n-skx-x710-64b-4t2c-base-ndr



2n-skx-xxv710

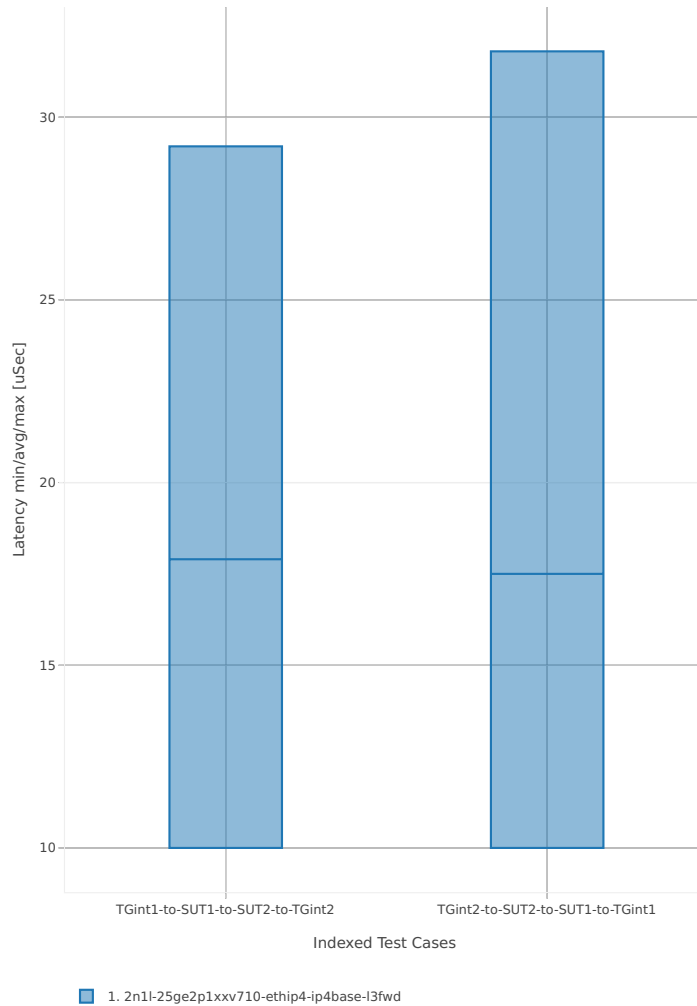
64b-2t1c-base

Latency: l3fwd-2n-skx-xxv710-64b-2t1c-base-ndr



64b-4t2c-base

Latency: l3fwd-2n-skx-xxv710-64b-4t2c-base-ndr



3.5 Comparisons

3.5.1 Current vs. Previous Release

Relative comparison of DPDK Testpmd and L3fwd packet throughput (NDR, PDR and MRR) between DPDK 18.05 and DPDK-18.02 (measured for CSIT-18.07 and CSIT-18.04 respectively) is calculated from results of tests running on 3-Node Intel Xeon Haswell testbeds (3n-hsw) in 1-core and 2-core configurations.

Listed mean and standard deviation values are computed based on a series of the same tests executed against respective DPDK releases to verify test results repeatability, with percentage change calculated for mean values.

Note: Test results have been generated by [FD.io test executor dpdk performance job 3n-hsw](#)¹²⁰ with RF result files csit-dpdk-perf-1807-*.zip [archived here](#).

NDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII 1t1c NDR comparison](#)
- [ASCII 2t2c NDR comparison](#)
- [CSV 1t1c NDR comparison](#)
- [CSV 2t2c NDR comparison](#)

PDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII 1t1c PDR comparison](#)
- [ASCII 2t2c PDR comparison](#)
- [CSV 1t1c PDR comparison](#)
- [CSV 2t2c PDR comparison](#)

3.5.2 3n-Skx vs. 3n-Hsw Testbeds

Relative comparison of DPDK 18.05 Testpmd and L3fwd packet throughput (NDR, PDR and MRR) is calculated for the same tests executed on 3-Node Skylake (3n-skx) and 3-Node Haswell (3n-hsw) physical testbed types, in 1-core, 2-core and 4-core configurations.

Note: Test results have been generated by [FD.io test executor dpdk performance job 3n-hsw](#)¹²¹ and [FD.io test executor dpdk performance job 3n-skx](#)¹²² with RF result files csit-dpdk-perf-1807-*.zip [archived here](#).

¹²⁰ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-hsw>

¹²¹ <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-hsw>

¹²² <https://jenkins.fd.io/view/csit/job/csit-dpdk-perf-verify-1807-3n-skx>

NDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII NDR comparison](#)
- [CSV NDR comparison](#)

PDR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII PDR comparison](#)
- [CSV PDR comparison](#)

MRR Comparison

Comparison tables in ASCII and CSV formats:

- [ASCII MRR comparison](#)
- [CSV MRR comparison](#)

3.6 Throughput Trending

In addition to reporting throughput comparison between DPDK releases, CSIT provides regular performance trending for DPDK release branches:

1. [Performance Dashboard](#)¹²³: per DPDK test case throughput trend, trend compliance and summary of detected anomalies.
2. [Trending Methodology](#)¹²⁴: throughput test metrics, trend calculations and anomaly classification (progression, regression).
3. [DPDK Trendline Graphs](#)¹²⁵: weekly DPDK Testpmd and L3fwd MRR throughput measurements against the trendline with anomaly highlights and associated CSIT test jobs.

¹²³ <https://docs.fd.io/csit/master/trending/introduction/index.html>

¹²⁴ <https://docs.fd.io/csit/master/trending/methodology/index.html>

¹²⁵ <https://docs.fd.io/csit/master/trending/trending/dpdk.html>

3.7 Test Environment

3.7.1 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, Arm, Intel Atom. More detailed description is provided in *Physical Testbeds* (page 4). Tested logical topologies are described in *Logical Topologies* (page 20).

3.7.2 Server Specifications

Complete technical specifications of compute servers used in CSIT physical testbeds are maintained on FD.io wiki pages: [CSIT/Testbeds: Xeon Hsw, VIRT](#)¹²⁶ and [CSIT Testbeds: Xeon Skx, Arm, Atom](#)¹²⁷.

3.7.3 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](#)¹²⁸.
2. Memory bandwidth – measure bandwidth with [Intel MLC tool](#)¹²⁹.
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 Intel Xeon Haswell and Intel Xeon Skylake testbeds.

3.7.4 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 [CSIT/Testbeds: Xeon Hsw, VIRT](#)¹³⁰.

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

¹²⁶ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Hsw,_VIRT.#FD.io_CSIT_testbeds_-_Xeon_Haswell.2C_VIRT

¹²⁷ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.#Server_Specification

¹²⁸ https://git.fd.io/pma_tools/tree/jitter

¹²⁹ <https://software.intel.com/en-us/articles/intelr-memory-latency-checker>

¹³⁰ https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Hsw,_VIRT.#FD.io_CSIT_testbeds_-_Xeon_Haswell.2C_VIRT

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.4.0-72-generic root=UUID=efb7e8b3-3548-4440-98f6-6ebe102e9ec6 ro isolcpus=1-
↳17,19-35 nohz_full=1-17,19-35 rcu_nocbs=1-17,19-35 intel_pstate=disable console=tty0
↳console=ttyS0,115200n8
```

Linux uname

```
$ uname -a
Linux t3-sut2 4.4.0-72-generic #93-Ubuntu SMP Fri Mar 31 14:07:41 UTC 2017 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 program will execute a dummy function 80000 times
Display is updated every 20000 displayUpdate intervals
Timings are in CPU Core cycles
Inst_Min:   Minimum Execution time during the display update interval(default is ~1 second)
Inst_Max:   Maximum Execution time during the display update interval(default is ~1 second)
Inst_jitter: Jitter in the Execution time during the display update interval. This is the value of
↳interest
last_Exec:  The Execution time of last iteration just before the display update
Abs_Min:   Absolute Minimum Execution time since the program started or statistics were reset
Abs_Max:   Absolute Maximum Execution time since the program started or statistics were reset
tmp:       Cumulative value calculated 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								
↳1	160024	172636	12612	160028	160024	172636	1573060608	3205463144
↳2	160024	188236	28212	160028	160024	188236	958595072	3205500844
↳3	160024	185676	25652	160028	160024	188236	344129536	3205485976
↳4	160024	172608	12584	160024	160024	188236	4024631296	3205472740
↳5	160024	179260	19236	160028	160024	188236	3410165760	3205502164
↳6	160024	172432	12408	160024	160024	188236	2795700224	3205452036
↳7	160024	178820	18796	160024	160024	188236	2181234688	3205455408
↳8	160024	172512	12488	160028	160024	188236	1566769152	3205461528
↳9	160024	172636	12612	160028	160024	188236	952303616	3205478820
↳10	160024	173676	13652	160028	160024	188236	337838080	3205470412
↳11	160024	178776	18752	160028	160024	188236	4018339840	3205481472
↳12	160024	172788	12764	160028	160024	188236	3403874304	3205492336

(continues on next page)

(continued from previous page)

	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
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
```

(continues on next page)

(continued from previous page)

```

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

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

```

(continues on next page)

(continued from previous page)

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  -
```

3.7.5 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 [CSIT Testbeds: Xeon Skx, Arm, Atom](#)¹³¹.

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

Linux cmdline

```
$ cat /proc/cmdline
BOOT_IMAGE=vmlinux-4.15.0-23-generic root=UUID=759ad671-ad46-441b-a75b-9f54e81837bb 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 use_off_
↪console=tty0 console=ttyS0,115200n8
131 https://wiki.fd.io/view/CSIT/Testbeds:_Xeon_Skx,_Arm,_Atom.#Server_Specification
```

(continued from previous page)

Linux uname

```
$ uname -a
Linux s5-t22-sut1 4.15.0-23-generic #25-Ubuntu SMP Wed May 23 18:02:16 UTC 2018 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 program 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 calcaulted 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								
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								

(continues on next page)

(continued from previous page)

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
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
```

(continues on next page)

(continued from previous page)

```
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
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  -
```

3.7.6 SUT Settings - Linux

System provisioning is done by combination of PXE boot unattended install and [Ansible](#)¹³² described in [CSIT Testbed Setup](#)¹³³.

Below a subset of the running configuration:

1. Xeon Haswell - Ubuntu 16.04.1 LTS

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.1 LTS
Release:       16.04
Codename:      xenial
```

2. Xeon Skylake - Ubuntu 18.04 LTS

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

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>

¹³² <https://www.ansible.com>

¹³³ <https://git.fd.io/csit/tree/resources/tools/testbed-setup/README.md?h=rls1807>

- **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.

Applied Boot Cmdline

1. Xeon Haswell - Ubuntu 16.04.1 LTS

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.4.0-72-generic root=UUID=35ea11e4-e44f-4f67-8cbe-12f09c49ed90 ro isolcpus=1-
↪17,19-35 nohz_full=1-17,19-35 rcu_nocbs=1-17,19-35 intel_pstate=disable console=tty0_
↪console=ttyS0,115200n8
```

2. Xeon Skylake - Ubuntu 18.04 LTS

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.15.0-23-generic root=UUID=3fa246fd-1b80-4361-bb90-f339a6bbed51 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
```

Host IRQ Affinity

IRQs are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ for l in `ls /proc/irq`; do echo 1 | sudo tee /proc/irq/$l/smp_affinity; done
```

Host RCU Affinity

RCUs are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ for i in `pgrep rcu[^c]`; do sudo taskset -pc 0 $i ; done
```

Host Writeback Affinity

Writebacks are pinned to core 0. The same configuration is applied in host Linux and guest VM.

```
$ echo 1 | sudo tee /sys/bus/workqueue/devices/writeback/cpumask
```

3.7.7 DUT Settings - DPDK

DPDK Version

DPDK 18.05

DPDK Compile Parameters

```
make install T=x86_64-native-linuxapp-gcc -j
```

Testpmd Startup Configuration

Testpmd startup configuration changes per test case with different settings for `$$CORES`, `$$RXQ` and `max-pkt-len` parameter if test is sending jumbo frames. Startup command template:

```
testpmd -c $$CORE_MASK -n 4 -- --numa --nb-ports=2 --portmask=0x3 --nb-cores=$$CORES --max-pkt-  
↪ len=9000 --txqflags=0 --forward-mode=io --rxq=$$RXQ --txq=$$TXQ --burst=64 --rxd=1024 --txd=1024 -  
↪ -disable-link-check --auto-start
```

L3FWD Startup Configuration

L3FWD startup configuration changes per test case with different settings for `$$CORES` and `enable-jumbo` parameter if test is sending jumbo frames. Startup command template:

```
l3fwd -l $$CORE_LIST -n 4 -- -P -L -p 0x3 --config='${port_config}' --enable-jumbo --max-pkt-  
↪ len=9000 --eth-dest=0,${adj_mac0} --eth-dest=1,${adj_mac1} --parse-ptype
```

3.7.8 TG Settings - TRex

TG Version

TRex v2.35

DPDK version

DPDK v17.11

TG Build Script used

TRex intallation¹³⁴

¹³⁴ https://git.fd.io/csit/tree/resources/tools/trex/trex_installer.sh?h=rls1807

TG Startup Configuration

```
$ cat /etc/trex_cfg.yaml
- port_limit      : 2
  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 --iom 0 > /tmp/trex.log 2>
->&1 &' > /dev/null
```

TG API Driver

TRex driver¹³⁵

3.8 Documentation

CSIT DPDK Performance Tests Documentation¹³⁶ contains detailed functional description and input parameters for each test case.

¹³⁵ https://git.fd.io/csit/tree/resources/tools/trex/trex_stateless_profile.py?h=rls1807

¹³⁶ <https://docs.fd.io/csit/rls1807/doc/tests.dpdk.perf.html>

4.1 Overview

4.1.1 Virtual Topologies

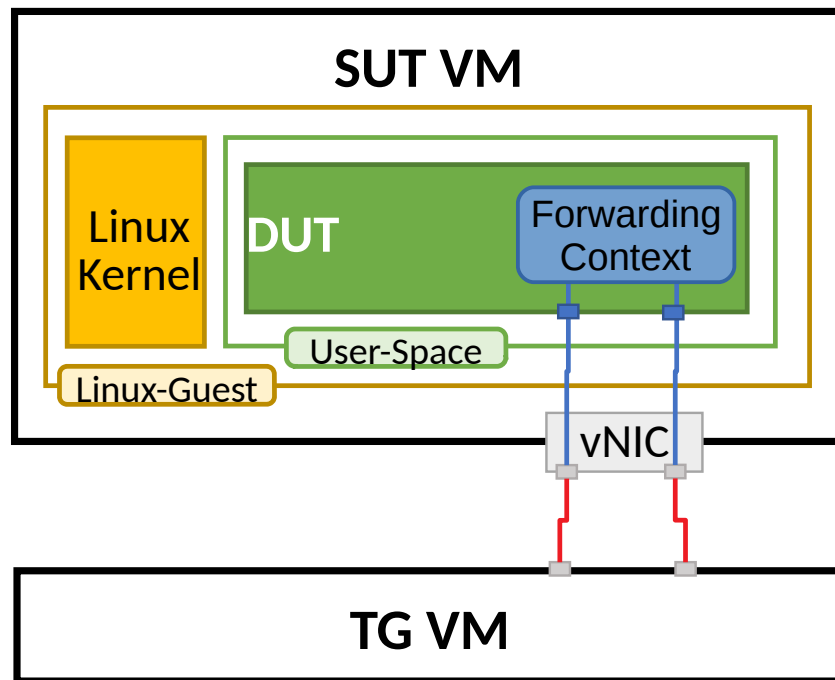
CSIT VPP functional tests are executed in VM-based virtual topologies created on demand using VIRL (Virtual Internet Routing Lab) simulation platform contributed by Cisco. VIRL runs on physical baremetal servers hosted by LF FD.io project. Based on the packet path thru SUT VMs, two distinct logical topology types are used for VPP DUT data plane testing:

1. vNIC-to-vNIC switching topologies.
2. Nested-VM service switching topologies.

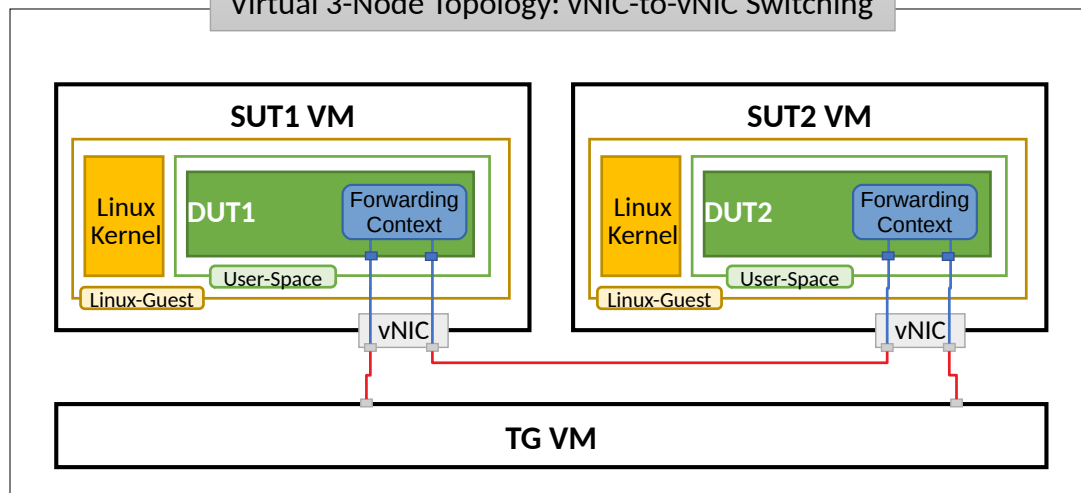
vNIC-to-vNIC Switching

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

Virtual 2-Node Topology: vNIC-to-vNIC Switching



Virtual 3-Node Topology: vNIC-to-vNIC Switching

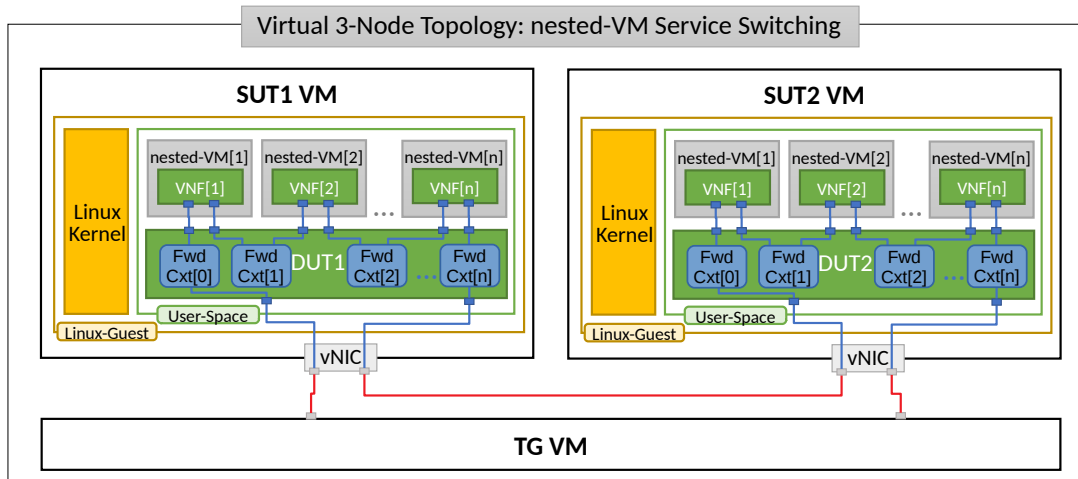


SUT1 and SUT2 are two VMs (running Ubuntu or Centos, depending on the test suite), TG is a Traffic Generator (running Ubuntu VM). SUTs run VPP SW application in Linux user-mode as a Device Under Test (DUT) within the VM. TG runs Scapy SW application as a packet Traffic Generator. Network connectivity between SUTs and to TG is provided using virtual NICs and VMs' virtio drivers.

Virtual testbeds 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 Virtual Machine and each connection that is drawn on the diagram is available for use in any test case. During the test execution, all nodes are reachable thru the Management network connected to every node via dedicated virtual NICs and virtual links (not shown above for clarity).

Nested-VM Service Switching

Nested-VM (Virtual Machine) service switching topology test cases require VPP DUT to communicate with nested-VM(s) over vhost-user virtual interfaces. Nested-VM(s) is(are) created on SUT1 and/or SUT2 for the duration of these particular test cases only. Virtual test topology with nested-VM(s) is shown in the figure below.



4.1.2 Functional Tests Coverage

CSIT-18.07 includes following VPP functionality tested in virtual VM 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.
DHCP	Dynamic Host Control Protocol Client and Proxy for IPv4 and IPv6 routing.
GRE	Generic Routing Encapsulation Overlay Tunnels for IPv4.
IPSec	IPSec tunnel and transport modes.
IPv4	IPv4 routing, RPF, ARP, Proxy ARP, ICMPv4.
IPv6	IPv6 routing, NS/ND, RA, ICMPv6.
L2BD	L2 Bridge-Domain switching for untagged Ethernet, dot1q and dot1ad tagged.
L2XC	L2 Cross-Connect switching for untagged Ethernet, dot1q and dot1ad tagged.
LISP	Locator/ID Separation Protocol overlay tunnels and locator/id mapping control.
QoS Policer Metering	Ingress packet rate metering and marking for IPv4, IPv6.
Softwire Tunnels	IPv4-in-IPv6 softwire tunnels.
Tap Interface	Baseline Linux tap interface tests.
IPFIX and SPAN	Telemetry IPFIX netflow statistics and SPAN port mirroring.
uRPF Source Security	Unicast Reverse Path Forwarding security for IPv4 and IPv6 routing.
VLAN Tag Translation	L2 VLAN tag translation 2to2, 2to1, 1to2, 1to1.
VRF Routing	Multi-context VRF IPVPN routing for IPv4 and IPv6.
VXLAN	VXLAN overlay tunneling for L2-over-IPv4 and -over-IPv6.

4.1.3 Functional Tests Naming

CSIT-18.07 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 541).

4.2 Release Notes

4.2.1 Changes in CSIT-18.07

1. CSIT FRAMEWORK

- Few test case bug fixes.

2. TEST CASE PORTING TO VPP_MAKE_TEST

- Implementation of VIRT functional integration tests in VPP_make_test.
- All VIRT tests identified as Priority-0 by FD.io VPP and CSIT projects have been ported to VPP_make_test. Detailed breakdown in [CSIT_VIRT migration progress](#)¹³⁷.

4.2.2 Known Issues

List of known issues in CSIT-18.07 for VPP functional tests in VIRT:

#	JiraID	Issue Description
1	CSIT-129 ¹³⁸ VPP-99 ¹³⁹	DHCPv4 client: Client responses to DHCPv4 OFFER sent with different XID. Client replies with DHCPv4 REQUEST message when received DHCPv4 OFFER message with different (wrong) XID.
2	CSIT-398 ¹⁴⁰ VPP-380 ¹⁴¹	Softwire - MAP-E: Incorrect calculation of IPv6 destination address when IPv4 prefix is 0. IPv6 destination address is wrongly calculated in case that IPv4 prefix is equal to 0 and IPv6 prefix is less than 40.
3	CSIT-399 ¹⁴² VPP-435 ¹⁴³	Softwire - MAP-E: Map domain is created when incorrect parameters provided. Map domain is created in case that the sum of suffix length of IPv4 prefix and PSID length is greater than EA bits length. IPv6 destination address contains bits written with PSID over the EA-bit length when IPv4 packet is sent.
4	CSIT-409 ¹⁴⁴ VPP-406 ¹⁴⁵	IPv6 RA: Incorrect IPv6 destination address in response to ICMPv6 Router Solicitation. Wrong IPv6 destination address (ff02::1) is used in ICMPv6 Router Advertisement packet sent as a response to received ICMPv6 Router Solicitation packet.
5	CSIT-565 ¹⁴⁶	Vhost-user: QEMU reconnect does not work. QEMU 2.5.0 used in CSIT does not support vhost-user reconnect. Requires upgrading CSIT VIRT environment to QEMU 2.7.0.

4.3 Test Environment

CSIT VPP functional tests are executed in FD.io VIRT testbeds. The physical VIRT testbed infrastructure consists of three VIRT servers:

¹³⁷ https://docs.google.com/spreadsheets/d/1PciV8XN9v1qHbIRUpFJoqyES29_vik7lcFDI73G1usc/edit?usp=sharing

¹³⁸ <https://jira.fd.io/browse/CSIT-129>

¹³⁹ <https://jira.fd.io/browse/VPP-99>

¹⁴⁰ <https://jira.fd.io/browse/CSIT-398>

¹⁴¹ <https://jira.fd.io/browse/VPP-380>

¹⁴² <https://jira.fd.io/browse/CSIT-399>

¹⁴³ <https://jira.fd.io/browse/VPP-435>

¹⁴⁴ <https://jira.fd.io/browse/CSIT-409>

¹⁴⁵ <https://jira.fd.io/browse/VPP-406>

¹⁴⁶ <https://jira.fd.io/browse/CSIT-565>

- tb4-virl1:
 - Status: Production
 - OS: Ubuntu 16.04.2
 - VIRL STD server version: 0.10.32.16
 - VIRL UWM server version: 0.10.32.16
- tb4-virl2:
 - Status: Production
 - OS: Ubuntu 16.04.2
 - VIRL STD server version: 0.10.32.16
 - VIRL UWM server version: 0.10.32.16
- tb4-virl3:
 - Status: Production
 - OS: Ubuntu 16.04.2
 - VIRL STD server version: 0.10.32.19
 - VIRL UWM server version: 0.10.32.19
- VIRL hosts: Cisco UCS C240-M4, each with 2x Intel Xeon E5-2699 v3 (2.30 GHz, 18c), 512GB RAM.

Whenever a patch is submitted to gerrit for review, parallel VIRL simulations are started to reduce the time of execution of all functional tests. The number of parallel VIRL simulations is equal to a number of test groups defined by TEST_GROUPS variable in `csit/bootstrap.sh` file. VIRL host to run VIRL simulation is selected based on least load algorithm per VIRL simulation.

Every VIRL simulation uses the same three-node logical ring topology - Traffic Generator (TG node) and two Systems Under Test (SUT1 and SUT2). The appropriate pre-built VPP packages built by Jenkins for the patch under review are then installed on the two SUTs, along with their `/etc/vpp/startup.conf` file, in all VIRL simulations.

4.3.1 SUT Settings - VIRL Guest VM

SUT VMs' settings are defined in [VIRL topologies directory](#)¹⁴⁷

- List of SUT VM interfaces:

```
<interface id="0" name="GigabitEthernet0/4/0"/> <interface id="1"
name="GigabitEthernet0/5/0"/> <interface id="2" name="GigabitEthernet0/6/0"/>
<interface id="3" name="GigabitEthernet0/7/0"/>
```
- Number of 2MB hugepages: 1024.
- Maximum number of memory map areas: 20000.
- Kernel Shared Memory Max: 2147483648 (`vm.nr_hugepages * 2 * 1024 * 1024`).

4.3.2 SUT Settings - VIRL Guest OS Linux

In CSIT terminology, the VM operating system for both SUTs that VPP-18.07 release has been tested with, is the following:

¹⁴⁷ <https://git.fd.io/csit/tree/resources/tools/virl/topologies/?h=rls1807>

1. Ubuntu VIRT image

This image implies Ubuntu 16.04.1 LTS, current as of yyyy-mm-dd (that is, package versions are those that would have been installed by a `apt-get update`, `apt-get upgrade` on that day), produced by CSIT disk image build scripts.

The exact list of installed packages and their versions (including the Linux kernel package version) are included in [VIRT ubuntu images lists](#)¹⁴⁸.

A replica of this VM image can be built by running the `build.sh` script in CSIT repository.

2. CentOS VIRT image

This image implies Centos 7.4-1711, current as of yyyy-mm-dd (that is, package versions are those that would have been installed by a `yum update`, `yum upgrade` on that day), produced by CSIT disk image build scripts.

The exact list of installed packages and their versions (including the Linux kernel package version) are included in [VIRT centos images lists](#)¹⁴⁹.

A replica of this VM image can be built by running the `build.sh` script in CSIT repository.

3. Nested VM image

In addition to the “main” VM image, tests which require VPP to communicate to a VM over a vhost-user interface, utilize a “nested” VM image.

This “nested” VM is dynamically created and destroyed as part of a test case, and therefore the “nested” VM image is optimized to be small, lightweight and have a short boot time. The “nested” VM image is not built around any established Linux distribution, but is based on [BuildRoot](#)¹⁵⁰, a tool for building embedded Linux systems. Just as for the “main” image, scripts to produce an identical replica of the “nested” image are included in CSIT GIT repository, and the image can be rebuilt using the “build.sh” script at [VIRT nested](#)¹⁵¹.

4.3.3 DUT Settings - VPP

Every System Under Test runs VPP SW application in Linux user-mode as a Device Under Test (DUT) node.

DUT Port Configuration

Port configuration of DUTs is defined in topology file that is generated per VIRT simulation based on the definition stored in [VIRT topologies directory](#)¹⁵².

Example of DUT nodes configuration:

DUT1:

type: DUT host: “10.30.51.157” port: 22 username: cisco honeycomb:

user: admin passwd: admin port: 8183 netconf_port: 2831

```
priv_key: | --BEGIN RSA PRIVATE KEY-- MIIePgIBAAKCAQEAWUDIT-
pzSHpwLQotZOFs4AgcPNEWCnP1AB2hWFmvl+8Kah/gb
v8ruZU9RqhPs56tyKzxbhvNkY4VbH5F1GilHZu3mLqzM4KfghMmaeMEjO1T7BYyd
vuBfTvluljfQ2vAlnYrDwn+ClxJk81mOpDgvrLEX4qVVh2sGh7UEkYy5r82DNa2
4VjzPB1J/c8a9zP8FoZUhYlzF4FLvRMjUADpbMXgJMsGpaZLmz95ap0Eot7vb1Cc
1LvF97iyBCrtIOSKRKA50ZhLgJMKmOwnYU+cP5718tbproDVi6VJOo7zeuXyetMs
8YBI9kWbIWG9BqP9jctFvsmi5G7hXgq1Y8u+DwiDAQABAoIBAQC/W4E0DHjLMny7
0bvW2YKzD0Zw3fttdB94tkm4PdZv5MybooPnsAvLaXVV0hEdfVi5kzSWNI/LY/tN
```

¹⁴⁸ <https://git.fd.io/csit/tree/resources/tools/disk-image-builder/ubuntu/lists/?h=rls1807>

¹⁴⁹ <https://git.fd.io/csit/tree/resources/tools/disk-image-builder/centos/lists/?h=rls1807>

¹⁵⁰ <https://buildroot.org/>

¹⁵¹ <https://git.fd.io/csit/tree/resources/tools/disk-image-builder/nested/?h=rls1807>

¹⁵² <https://git.fd.io/csit/tree/resources/tools/virt/topologies/?h=rls1807>

```

EP1BgGphc2QgB59/PPxGwFjDCvUzlsZpynBHe+B/qh5ExNQcVvsIOqWI7DXIXaN
Oi/khOzmJ6HncRRah1spKimYRsaUUDskyg7q3QqMwVaqBbbMvLs/w7ZWd/zoDqCU
MY/pCl6hkB3QbRo0OdiZLohphBI2ShABTwjvVyyKL5UA4jAEneJrhH5gWVLXnfgD
p62W5CollKEYblC8mUkPxpP7Qo277zw3xaq+oktIzhc5SUEUd7nJZtNqVAHqklitW
79VmpKyxAoGBAPfU+kqNPATsvp+x1n5sn2SgipzDtgi9QqNmC4cjrQQaaqI57SG
OHw1jX8i7L2G1WvVtkHg060nIEVo5n65ffFOqeVBezLVJ7ghWI8U+oBiJJyQ4boD
GJVnsoOSUQ0rtuGd9eVwfDk3ol9aCNOKK53oPfiYli29pyu4l095kg11AoGBAMef
bPEMBI/2XmCPshLSwhGFI+dW8d+Klluj3CUQ/0vUlvma3dfBOYNslwAgTP0iIUTg
8DYE6KBCdPtxAUEIOYAEAKB9ry1tKR2NQEIPfslYytKErtwjAiqSiOheM6+zwEzu
f54Z4oBhsMSL0jXoOMnu+NZzEc6EUdQeY4O+jhjzAoGBAlogC3dtjMPGKTP7+93u
UE/Xliol8fWg9fj3sMka4IMu+pVvRCRbAjRH7JrFLkjbUyuMqs3Arnk9K+gbdQt/
+m95Njtt6WoFXuPCwgbM3GidSmZwYT4454SfDzVBYScedCNm1FuR+8ov9bFLDtGT
D4gsngnGJj1MDFXTxZEn4nzZAoGBAKCg4WmpUPaCuXibyB+rZavxwsTNSn2IJ83/
sYJGBhf/raiV/FLDUcM1vYg5dZnu37RsB/5/vqxOLZGyYd7x+Jo5HkQGPNKgNwhn
g8BkdZIRF8uEJqxOo0ycdOU7n/2O93swlpKWo5LliRPuqqzj+uZKnAL7vuVdxfaY
qVz2daMPAoGBALgaaKa3voU/HO1PYLWlhFrBThyJ+BQSQ8OqrEzC8AnegWfXRAM8
EqrzZXI7ACUuo1dH0Eipm41j2+BZWIQjiUgq5uj8+zyz+EU1ZRRyJcOKzbDACEuD
BpWWSXGBI5G4CppeYLjMUHZpJYeX1USULJQd2c4crLJKb76E8gz3Z9kN
--END RSA PRIVATE KEY--

```

interfaces:

port1: mac_address: "fa:16:3e:9b:89:52" pci_address: "0000:00:04.0" link:
link1

port2: mac_address: "fa:16:3e:7a:33:60" pci_address: "0000:00:05.0" link:
link4

port3: mac_address: "fa:16:3e:29:b7:ae" pci_address: "0000:00:06.0" link:
link3

port4: mac_address: "fa:16:3e:76:8d:ff" pci_address: "0000:00:07.0" link:
link6

DUT2: type: DUT host: "10.30.51.156" port: 22 username: cisco honeycomb:

user: admin passwd: admin port: 8183 netconf_port: 2831

```

priv_key: | --BEGIN RSA PRIVATE KEY-- MIIEpgIBAAKCAQEAwUDIT-
pzSHpwLQotZOFs4AgcPNEwCnP1AB2hWfMvI+8Kah/gb
v8ruZU9RqhPs56tyKzxbhvNkY4VbH5F1GihHZu3mLqzM4KfghMmaeMEjO1T7BYyd
vuBfTvluljfq2vAlnYrDwn+ClxJk81m0pDgvrLEX4qVVh2sGh7UEkYy5r82DNa2
4VjzPB1J/c8a9zP8FoZUhYlzF4FLvRMjUADpbMXgJMsGpaZLmz95apOEot7vb1Cc
1LvF97iyBCrtIOSKRKA50ZhlGjMKmOwnYU+cP5718tbproDV16VJOo7zeuXyetMs
8YBI9kWBWlWG9BqP9jctFvsmi5G7hXgq1Y8u+DwIDAQABAoIBAQC/W4E0DHJLMny7
0bvw2YKzD0Zw3fttdB94tkm4PdZv5MybooPnsAvLaXVV0hEdfvi5kzSWNI/LY/tN
EP1BgGphc2QgB59/PPxGwFjDCvUzlsZpynBHe+B/qh5ExNQcVvsIOqWI7DXIXaN
Oi/khOzmJ6HncRRah1spKimYRsaUUDskyg7q3QqMwVaqBbbMvLs/w7ZWd/zoDqCU
MY/pCl6hkB3QbRo0OdiZLohphBI2ShABTwjvVyyKL5UA4jAEneJrhH5gWVLXnfgD
p62W5CollKEYblC8mUkPxpP7Qo277zw3xaq+oktIzhc5SUEUd7nJZtNqVAHqklitW
79VmpKyxAoGBAPfU+kqNPATsvp+x1n5sn2SgipzDtgi9QqNmC4cjrQQaaqI57SG
OHw1jX8i7L2G1WvVtkHg060nIEVo5n65ffFOqeVBezLVJ7ghWI8U+oBiJJyQ4boD
GJVnsoOSUQ0rtuGd9eVwfDk3ol9aCNOKK53oPfiYli29pyu4l095kg11AoGBAMef
bPEMBI/2XmCPshLSwhGFI+dW8d+Klluj3CUQ/0vUlvma3dfBOYNslwAgTP0iIUTg
8DYE6KBCdPtxAUEIOYAEAKB9ry1tKR2NQEIPfslYytKErtwjAiqSiOheM6+zwEzu
f54Z4oBhsMSL0jXoOMnu+NZzEc6EUdQeY4O+jhjzAoGBAlogC3dtjMPGKTP7+93u
UE/Xliol8fWg9fj3sMka4IMu+pVvRCRbAjRH7JrFLkjbUyuMqs3Arnk9K+gbdQt/
+m95Njtt6WoFXuPCwgbM3GidSmZwYT4454SfDzVBYScedCNm1FuR+8ov9bFLDtGT
D4gsngnGJj1MDFXTxZEn4nzZAoGBAKCg4WmpUPaCuXibyB+rZavxwsTNSn2IJ83/
sYJGBhf/raiV/FLDUcM1vYg5dZnu37RsB/5/vqxOLZGyYd7x+Jo5HkQGPNKgNwhn
g8BkdZIRF8uEJqxOo0ycdOU7n/2O93swlpKWo5LliRPuqqzj+uZKnAL7vuVdxfaY
qVz2daMPAoGBALgaaKa3voU/HO1PYLWlhFrBThyJ+BQSQ8OqrEzC8AnegWfXRAM8

```

```

EqrzZXI7ACUuo1dH0Eipm41j2+BZWIQjiUgq5uj8+yzy+EU1ZRRyJcOKzbDACeuD
BpWWSXGBI5G4CppeYLjMUHZpJYeX1USULJQd2c4crLJKb76E8gz3Z9kN
--END RSA PRIVATE KEY--

```

interfaces:

port1: mac_address: "fa:16:3e:ad:6c:7d" pci_address: "0000:00:04.0" link: link2

port2: mac_address: "fa:16:3e:94:a4:99" pci_address: "0000:00:05.0" link: link5

port3: mac_address: "fa:16:3e:75:92:da" pci_address: "0000:00:06.0" link: link3

port4: mac_address: "fa:16:3e:2c:b1:2a" pci_address: "0000:00:07.0" link: link6

VPP Version

VPP-18.07 release

VPP Installed Packages - Ubuntu

```

$ dpkg -l vpp\*
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version          Architecture Description
+++-----
ii vpp              18.07-release    amd64          ↵
↳Vector Packet Processing--executables
ii vpp-dbg          18.07-release    amd64          ↵
↳Vector Packet Processing--debug symbols
ii vpp-dev          18.07-release    amd64          ↵
↳Vector Packet Processing--development support
ii vpp-dpdk-dkms    18.05-vpp2       amd64          ↵
↳DPDK Development Package for VPP - Kernel Modules
ii vpp-lib          18.07-release    amd64          ↵
↳Vector Packet Processing--runtime libraries
ii vpp-plugins      18.07-release    amd64          ↵
↳Vector Packet Processing--runtime plugins

```

VPP Installed Packages - Centos

```

$ rpm -qai vpp*
Name       : vpp
Version    : 18.07
Release    : release
Architecture: x86_64
Install Date: Tue 31 Jul 2018 02:59:45 AM EDT
Group      : Unspecified
Size       : 2396993
License    : ASL 2.0
Signature  : (none)
Source RPM : vpp-18.07-release.src.rpm
Build Date : Mon 30 Jul 2018 08:20:19 PM EDT
Build Host : c3de88e7d43c
Relocations : (not relocatable)
Summary    : Vector Packet Processing
Description :

```

(continues on next page)

(continued from previous page)

```
This package provides VPP executables: vpp, vpp_api_test, vpp_json_test
vpp - the vector packet engine
vpp_api_test - vector packet engine API test tool
vpp_json_test - vector packet engine JSON test tool
Name       : vpp-lib
Version    : 18.07
Release    : release
Architecture: x86_64
Install Date: Tue 31 Jul 2018 02:59:45 AM EDT
Group      : System Environment/Libraries
Size       : 27134058
License    : ASL 2.0
Signature  : (none)
Source RPM : vpp-18.07-release.src.rpm
Build Date : Mon 30 Jul 2018 08:20:19 PM EDT
Build Host : c3de88e7d43c
Relocations : (not relocatable)
Summary    : VPP libraries
Description :
This package contains the VPP shared libraries, including:
vppinfra - foundation library supporting vectors, hashes, bitmaps, pools, and string formatting.
svm - vm library
vlib - vector processing library
vlib-api - binary API library
vnet - network stack library
Name       : vpp-selinux-policy
Version    : 18.07
Release    : release
Architecture: x86_64
Install Date: Tue 31 Jul 2018 02:59:44 AM EDT
Group      : System Environment/Base
Size       : 86709
License    : ASL 2.0
Signature  : (none)
Source RPM : vpp-18.07-release.src.rpm
Build Date : Mon 30 Jul 2018 08:20:19 PM EDT
Build Host : c3de88e7d43c
Relocations : (not relocatable)
Summary    : VPP Security-Enhanced Linux (SELinux) policy
Description :
This package contains a tailored VPP SELinux policy
Name       : vpp-devel
Version    : 18.07
Release    : release
Architecture: x86_64
Install Date: Tue 31 Jul 2018 02:59:47 AM EDT
Group      : Development/Libraries
Size       : 11452203
License    : ASL 2.0
Signature  : (none)
Source RPM : vpp-18.07-release.src.rpm
Build Date : Mon 30 Jul 2018 08:20:19 PM EDT
Build Host : c3de88e7d43c
Relocations : (not relocatable)
Summary    : VPP header files, static libraries
Description :
This package contains the header files for VPP.
Install this package if you want to write a
program for compilation and linking with vpp lib.
vlib
vlibmemory
```

(continues on next page)

(continued from previous page)

```
vnet - devices, classify, dhcp, ethernet flow, gre, ip, etc.
vpp-api
vppinfra
Name       : vpp-plugins
Version    : 18.07
Release    : release
Architecture: x86_64
Install Date: Tue 31 Jul 2018 02:59:47 AM EDT
Group      : System Environment/Libraries
Size       : 52282610
License    : ASL 2.0
Signature  : (none)
Source RPM : vpp-18.07-release.src.rpm
Build Date : Mon 30 Jul 2018 08:20:19 PM EDT
Build Host : c3de88e7d43c
Relocations : (not relocatable)
Summary    : Vector Packet Processing--runtime plugins
Description :
This package contains VPP plugins
```

VPP Startup Configuration

VPP startup configuration is common for all test cases except test cases related to SW Crypto device.

Common Configuration

```
$ cat /etc/vpp/startup.conf
unix {
    nodaemon
    log /var/log/vpp/vpp.log
    full-coredump
    cli-listen /run/vpp/cli.sock
    gid vpp
}

api-trace {
## This stanza controls binary API tracing. Unless there is a very strong reason,
## please leave this feature enabled.
    on
## Additional parameters:
##
## To set the number of binary API trace records in the circular buffer, configure nitems
##
## nitems <nnn>
##
## To save the api message table decode tables, configure a filename. Results in /tmp/<filename>
## Very handy for understanding api message changes between versions, identifying missing
## plugins, and so forth.
##
## save-api-table <filename>
}

api-segment {
    gid vpp
}

socksvr {
    default
}
```

(continues on next page)

(continued from previous page)

```
cpu {
  ## In the VPP there is one main thread and optionally the user can create worker(s)
  ## The main thread and worker thread(s) can be pinned to CPU core(s) manually or automatically

  ## Manual pinning of thread(s) to CPU core(s)

  ## Set logical CPU core where main thread runs, if main core is not set
  ## VPP will use core 1 if available
  # main-core 1

  ## Set logical CPU core(s) where worker threads are running
  # corelist-workers 2-3,18-19

  ## Automatic pinning of thread(s) to CPU core(s)

  ## Sets number of CPU core(s) to be skipped (1 ... N-1)
  ## Skipped CPU core(s) are not used for pinning main thread and working thread(s).
  ## The main thread is automatically pinned to the first available CPU core and worker(s)
  ## are pinned to next free CPU core(s) after core assigned to main thread
  # skip-cores 4

  ## Specify a number of workers to be created
  ## Workers are pinned to N consecutive CPU cores while skipping "skip-cores" CPU core(s)
  ## and main thread's CPU core
  # workers 2

  ## Set scheduling policy and priority of main and worker threads

  ## Scheduling policy options are: other (SCHED_OTHER), batch (SCHED_BATCH)
  ## idle (SCHED_IDLE), fifo (SCHED_FIFO), rr (SCHED_RR)
  # scheduler-policy fifo

  ## Scheduling priority is used only for "real-time policies (fifo and rr),
  ## and has to be in the range of priorities supported for a particular policy
  # scheduler-priority 50
}

# dpdk {
  ## Change default settings for all interfaces
  # dev default {
    ## Number of receive queues, enables RSS
    ## Default is 1
    # num-rx-queues 3

    ## Number of transmit queues, Default is equal
    ## to number of worker threads or 1 if no workers threads
    # num-tx-queues 3

    ## Number of descriptors in transmit and receive rings
    ## increasing or reducing number can impact performance
    ## Default is 1024 for both rx and tx
    # num-rx-desc 512
    # num-tx-desc 512

    ## VLAN strip offload mode for interface
    ## Default is off
    # vlan-strip-offload on
  # }

  ## Whitelist specific interface by specifying PCI address
  # dev 0000:02:00.0
```

(continues on next page)

(continued from previous page)

```

## Whitelist specific interface by specifying PCI address and in
## addition specify custom parameters for this interface
# dev 0000:02:00.1 {
#     num-rx-queues 2
# }

## Specify bonded interface and its slaves via PCI addresses
##
## Bonded interface in XOR load balance mode (mode 2) with L3 and L4 headers
# vdev eth_bond0,mode=2,slave=0000:02:00.0,slave=0000:03:00.0,xmit_policy=134
# vdev eth_bond1,mode=2,slave=0000:02:00.1,slave=0000:03:00.1,xmit_policy=134
##
## Bonded interface in Active-Back up mode (mode 1)
# vdev eth_bond0,mode=1,slave=0000:02:00.0,slave=0000:03:00.0
# vdev eth_bond1,mode=1,slave=0000:02:00.1,slave=0000:03:00.1

## Change UIO driver used by VPP, Options are: igb_uio, vfio-pci,
## uio_pci_generic or auto (default)
# uio-driver vfio-pci

## Disable mutli-segment buffers, improves performance but
## disables Jumbo MTU support
# no-multi-seg

## Increase number of buffers allocated, needed only in scenarios with
## large number of interfaces and worker threads. Value is per CPU socket.
## Default is 16384
# num-mbufs 128000

## Change hugepages allocation per-socket, needed only if there is need for
## larger number of mbufs. Default is 256M on each detected CPU socket
# socket-mem 2048,2048

## Disables UDP / TCP TX checksum offload. Typically needed for use
## faster vector PMDs (together with no-multi-seg)
# no-tx-checksum-offload
# }

# plugins {
## Adjusting the plugin path depending on where the VPP plugins are
#     path /home/bms/vpp/build-root/install-vpp-native/vpp/lib64/vpp_plugins

## Disable all plugins by default and then selectively enable specific plugins
# plugin default { disable }
# plugin dpdk_plugin.so { enable }
# plugin acl_plugin.so { enable }

## Enable all plugins by default and then selectively disable specific plugins
# plugin dpdk_plugin.so { disable }
# plugin acl_plugin.so { disable }
# }

## Alternate syntax to choose plugin path
# plugin_path /home/bms/vpp/build-root/install-vpp-native/vpp/lib64/vpp_plugins

```

SW Crypto Device Configuration

```

$ cat /etc/vpp/startup.conf
unix

```

(continues on next page)

(continued from previous page)

```

{
  cli-listen /run/vpp/cli.sock
  gid vpp
  nodaemon
  full-coredump
  log /tmp/vpp.log
}
api-segment
{
  gid vpp
}
dpdk
{
  vdev cryptodev_aesni_gcm_pmd,socket_id=0
  vdev cryptodev_aesni_mb_pmd,socket_id=0
}

```

4.3.4 TG Settings - Scapy

Traffic Generator node is VM running the same OS Linux as SUTs. Ports of this VM are used as source (Tx) and destination (Rx) ports for the traffic.

Traffic scripts of test cases are executed on this VM.

TG VM Configuration

Configuration of the TG VMs is defined in [VIRL topologies directory](#)¹⁵³.

/csit/resources/tools/virl/topologies/double-ring-nested.xenial.virl

- List of TG VM interfaces::

```

<interface id="0" name="eth1"/>
<interface id="1" name="eth2"/>
<interface id="2" name="eth3"/>
<interface id="3" name="eth4"/>
<interface id="4" name="eth5"/>
<interface id="5" name="eth6"/>

```

TG Port Configuration

Port configuration of TG is defined in topology file that is generated per VIRL simulation based on the definition stored in [VIRL topologies directory](#)¹⁵⁴.

Example of TG node configuration::

```

TG:
  type: TG
  host: "10.30.51.155"
  port: 22
  username: cisco
  priv_key: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIIEpgIBAAKCAQEAwUD1TpzSHpwLQotZOFs4AgcPNEwCnP1AB2hWFmvI+8Kah/gb
    v8ruZU9RqhPs56tyKzxbhvNkY4VbH5F1G1lHZu3mLqzM4KfghMmaeMEjO1T7BYyd

```

(continues on next page)

¹⁵³ <https://git.fd.io/csit/tree/resources/tools/virl/topologies/?h=rls1807>

¹⁵⁴ <https://git.fd.io/csit/tree/resources/tools/virl/topologies/?h=rls1807>

(continued from previous page)

```

vuBfTvIlu1jfQ2vAlnYrDwn+C1xJk81m0pDgvrLEX4qVvh2sGh7UEkYy5r82DNa2
4VjzPB1J/c8a9zP8FoZuHYIZf4FLvRMjUADpbMXgJMsGpaZLmz95ap0Eot7vb1Cc
1LvF97iyBCrtIOSKRKA50ZHLGjMKmOwnYU+cP5718tbproDVi6VJ0o7zeuXyetMs
8YB19kWB1WG9BqP9jctFvsmi5G7hXgq1Y8u+DwIDAQABAOIBAQC/W4E0DHjLMny7
0bvW2YkZD0Zw3fttdB94tkm4PdZv5MybooPnsAvLaXVV0hEdfVi5kzSWN1/LY/tN
EP1BgGphc2QgB59/PPxGwFIjDCvUz1sZpynBHe+B/qh5ExNqCvvsIQqWI7DX1XaN
0i/kh0zmJ6HncRRah1spKimYRsaUUDskyg7q3QqMWVaqBbbMvLs/w7ZWd/zoDqCU
MY/pCI6hkB3QbRo00diZLohphB12ShABTjvVyyKL5UA4jAeneJrhH5gWVLXnfgD
p62W5CollKEYb1C8mUkPxpP7Qo277zw3xaq+oktIZhc5SUEUd7nJZtNqVAHqkItW
79VmpKyxAoGBAPfU+kqNPaTSvp+x1n5sn2SgipzDgtgi9QqNmC4cjtrQqaaqI57SG
OHw1jX8i7L2G1WvVtkHg060n1EVo5n65ffF0qeVBezLVJ7ghWI8U+oBiJJyQ4boD
GJVNs0SUQ0rtuGd9eVwfdk3o19aCN0KK53oPfIYli29pyu4l095kg11AoGBAMef
bPEMBI/2XmCPshLSwhGF1+dW8d+K1luj3CUQ/0vUlvma3dfBOYNsIwAgTP0iIUTg
8DYE6KBCdPtXAEI0YAEAKB9ry1tkR2NQEIPfs1YytKErtwjAiqSi0heM6+zwEzu
f54Z4oBhMSL0jXoMnu+NZZEc6EUdQeY40+jhJzAoGBAIOgc3dtjMPGKTP7+93u
UE/XIioI8fWg9fj3sMka4IMu+pVvRCRBAjRH7JrFLkjbUyuMqs3Arnk9K+gbdQt/
+m95Njtt6WofXUPcwgBm3GidSmZwYT4454SfDzVBYScedCNm1FuR+8ov9bFLDtGT
D4gsngnGjJ1MDFXTxZEn4nzZAoGBAKCg4WmpUPaCuXibyB+rZavxwsTNSn2lJ83/
sYJGBhf/raiV/FLDUcM1vYg5dZnu37RsB/5/vqxOLZGyYd7x+Jo5HkQGPnKgNwhn
g8BkdZIRF8uEJqx0o0yycd0U7n/2093swIpKWo5LIiRPuqqzj+uZKnAL7vuVdxfaY
qVz2daMPAoGBALgaaKa3voU/H01PYLWIhFrBThyJ+BQSQ80qrEzC8AnegWFxRAM8
EqrzZX17ACUuo1dH0Eipm41j2+BZW1QjiUgq5uj8+zyz+EU1ZRRyJcOKzbdACeuD
BpWWSXGBI5G4CppeYLjMUHJpJYeX1USULJQd2c4crLJKb76E8gz3Z9kN
-----END RSA PRIVATE KEY-----

```

interfaces:

```

port3:
  mac_address: "fa:16:3e:b9:e1:27"
  pci_address: "0000:00:06.0"
  link: link1
  driver: virtio-pci
port4:
  mac_address: "fa:16:3e:e9:c8:68"
  pci_address: "0000:00:07.0"
  link: link4
  driver: virtio-pci
port5:
  mac_address: "fa:16:3e:e8:d3:47"
  pci_address: "0000:00:08.0"
  link: link2
  driver: virtio-pci
port6:
  mac_address: "fa:16:3e:cf:ca:58"
  pci_address: "0000:00:09.0"
  link: link5
  driver: virtio-pci

```

Traffic Generator

Functional tests utilize Scapy as a traffic generator. Scapy v2.3.1 is used for VPP-18.07 release tests.

4.4 Documentation

CSIT VPP Functional Tests Documentation¹⁵⁵ contains detailed functional description and input parameters for each test case.

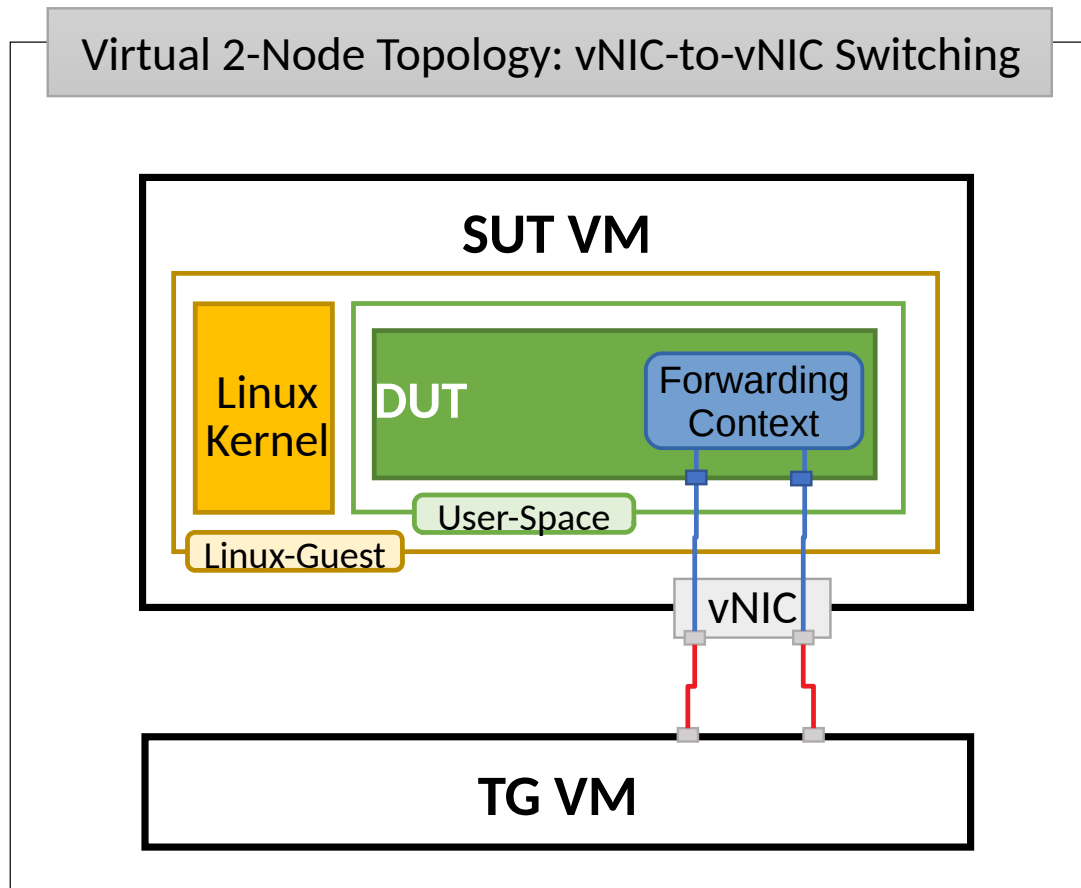
¹⁵⁵ <https://docs.fd.io/csit/rls1807/doc/tests.vpp.func.html>

5.1 Overview

5.1.1 Virtual Topologies

CSIT HoneyComb functional tests are executed in VM-based virtual topologies created on demand using VIRT simulation platform contributed by Cisco. VIRT runs on physical baremetal servers hosted by LF FD.io project.

All tests are executed in two-node virtual test topology shown in the figure below.



SUT (System Under Test) is a VM running Ubuntu Linux (or Centos, depending on the test suite), TG (Traffic Generator) is another VM running Ubuntu Linux. SUT VMs run HoneyComb management agent and VPP in Linux user-mode as a combined DUT (Device Under Test). TG runs Scapy application as a packet Traffic Generator. Virtual connectivity between SUT and TG is provided using virtual NICs using VMs' virtio drivers.

5.1.2 Functional Tests Coverage

CSIT-18.07 includes following HoneyComb functionality tested in virtual VM environment:

Functionality	Description
ACL	<ul style="list-style-type: none"> • CRD for low-level classifiers: table and session management, interface assignment. • Configure up to 2 classify tables. • Configure up to 2 classify sessions on one table. • Assign classify session to a physical interface. • Remove tables, sessions, interface assignments. • Test case count: 9.
ACL-PLUGIN	<ul style="list-style-type: none"> • CRD for high-level classifier. • MAC + IP address classification. • IPv4, IPv6 address classification. • TCP, UDP, ICMP, ICMPv6 protocol and next-header classification. • port number classification. • ICMP, ICMPv6 code and type classification. • Test case count: 15.
Basic interface management	<ul style="list-style-type: none"> • CRUD for interface state. • ipv4/ipv6 address, ipv4 neighbor, MTU value. • Test case count: 14.
Border Gateway Protocol	<ul style="list-style-type: none"> • CRUD and functional tests for BGP. • Configure peers and routes • Check interactions with another BGP peer. • Test case count: 13.
DHCP Relay	<ul style="list-style-type: none"> • CRD for DHCP relay feature. • Configure DHCP Relays. • IPv4 and IPv6 variants. • Test case count: 4.
Honeycomb Infrastructure	<ul style="list-style-type: none"> • Configuration persistence. • Netconf notifications for interface events. • Netconf negative tests aimed at specific issues. • Netconf/Restconf northbound over IPv6. • Test case count: 12.
L2BD	<ul style="list-style-type: none"> • CRUD for L2 Bridge-Domain, interface assignment. • Create up to two bridge domains with all implemented functions turned on: flooding, unknown-unicast flooding, forwarding, learning, arp-termination. • Assign up to two physical interfaces to a single bridge domain. • Remove interface assignments, remove bridge domains. • Test case count: 5.
580 FIB	<p style="text-align: center;">Chapter 5. HoneyComb Functional</p> <ul style="list-style-type: none"> • CRD for L2-FIB entries. • Create 4 FIB entries: one of each for filter/forward, static/dynamic combinations. • Remove FIB entries

Total 219 Honeycomb functional tests in the CSIT-18.07.

Operational data in Honeycomb should mirror configuration data at all times. Because of this, test cases follow this general pattern:

1. read operational data of the feature using restconf.
2. read status of the feature using VPP API dump.
3. modify configuration of the feature using restconf.
4. verify changes to operational data using restconf.
5. verify changes using VPP API dump, OR
6. send a packet to VPP node and observe behaviour to verify configuration.

Test cases involving network interfaces utilize the first two interfaces on the DUT node.

5.1.3 Functional Tests Naming

CSIT-18.07 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 541).

5.2 Release Notes

5.2.1 Changes in CSIT-18.07

No changes.

5.2.2 Known Issues

Here is the list of known issues in CSIT-18.07 for Honeycomb functional tests in VIRL:

#	JiraID	Issue Description
1	HONEYCOMB-403 ¹⁵⁶	IPv6 BGP route configuration. Configuring IPv6 route results in missing writer for IPv6Route and IPv6NextHop exception.
2	VPP-649 ¹⁵⁷	When configuring two IP addresses from the same subnet on an interface, VPP refuses the configuration, but returns code 200:OK. This can cause desync between Honeycomb's config and operational data.
3	VPP-875 ¹⁵⁸	VxLAN GPE configuration crashes VPP. Specific VxLAN GPE configurations cause VPP to crash and restart.
4	HC2VPP-254 ¹⁵⁹	Operational data for IPv6 special routes. Special hop routes are misidentified as regular routes in operational data.
5	HC2VPP-263 ¹⁶⁰	LISP PITR feature configuration. Locator set reference in operational data is incorrect.
6	CSIT-1210 ¹⁶¹	Unnumbered interface configuration. VPP does not send IP addresses for unnumbered interfaces anymore. HC CSIT tests were relying on that.

¹⁵⁶ <https://jira.fd.io/browse/HONEYCOMB-403>

¹⁵⁷ <https://jira.fd.io/browse/VPP-649>

¹⁵⁸ <https://jira.fd.io/browse/VPP-875>

¹⁵⁹ <https://jira.fd.io/browse/HC2VPP-254>

¹⁶⁰ <https://jira.fd.io/browse/HC2VPP-263>

¹⁶¹ <https://jira.fd.io/browse/CSIT-1210>

5.3 Test Environment

FD.io CSIT HoneyComb functional tests are executed in the same FD.io virtual environment as used CSIT VPP functional tests. See description in [VPP Functional Tests Environment](#).

5.4 Documentation

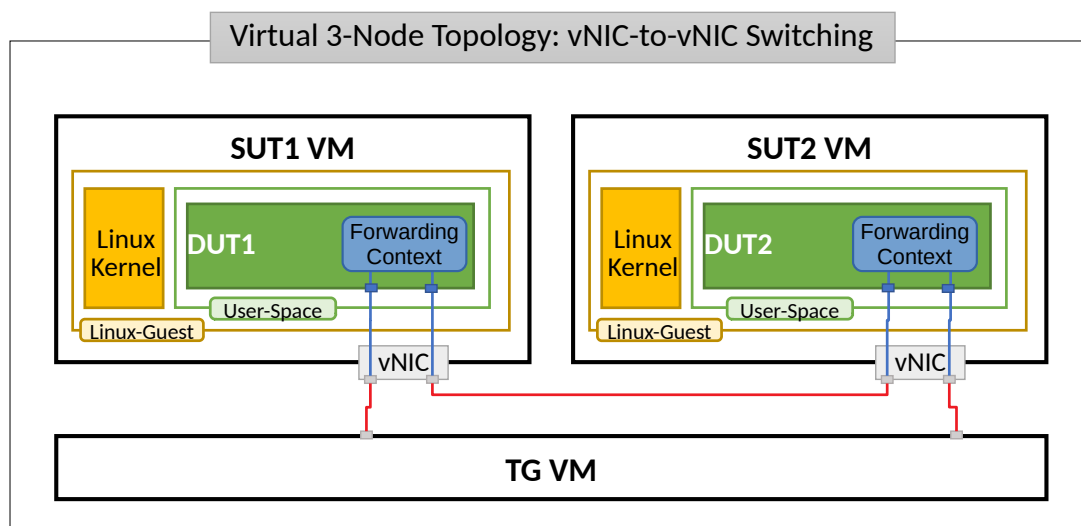
[CSIT Honeycomb Functional Tests Documentation](#)¹⁶² contains detailed functional description and input parameters for each test case.

¹⁶² <https://docs.fd.io/csit/rls1807/doc/tests.vpp.func.honeycomb.html>

6.1 Overview

6.1.1 Virtual Topologies

CSIT NSH_SFC functional tests are executed in VM-based virtual topologies created on demand using VIRT simulation platform contributed by Cisco. VIRT runs on physical baremetal servers hosted by LF FD.io project. All tests are executed in three-node virtual test topology shown in the figure below.



SUT1 and SUT2 are two VMs (running Ubuntu or Centos, depending on the test suite), TG is a Traffic Generator (running Ubuntu VM). SUTs run VPP with nsh-plugin in Linux user-mode as a Device Under Test (DUT) within the VM. TG runs Scapy SW application as a packet Traffic Generator. Network connectivity between SUTs and to TG is provided using virtual NICs and VMs' virtio drivers.

6.1.2 Functional Tests Coverage

CSIT-18.07 includes following NSH_SFC functionality tested in virtual VM environment:

Functionality	Description
NSH_SFC Classifier	TG sends some TCP packets to test NSH_SFC Classifier functional. DUT1 will receive these packets from one NIC and loopback the VXLAN-GPE-NSH encapsulated packets to the TG from other NIC. Test case count: 7.
NSH_SFC Proxy Inbound	TG sends some VXLAN-GPE-NSH encapsulated packets to test NSH_SFC Proxy Inbound functional. DUT1 will receive these packets from one NIC and loopback the VXLAN encapsulated packets to the TG from other NIC. Test case count: 6.
NSH_SFC Proxy Outbound	TG sends some VXLAN encapsulated packets to test NSH_SFC Proxy Outbound functional. DUT1 will receive these packets from one NIC and loopback the VXLAN-GPE-NSH encapsulated packets to the TG from other NIC. Test case count: 6.
NSH_SFC Service Function Forward	TG sends some VXLAN-GPE-NSH encapsulated packets to test NSH_SFC Service Function Forward functional. DUT1 will receive these packets from one NIC and swap the VXLAN-GPE-NSH header, after that DUT1 loopback the VXLAN-GPE-NSH encapsulated packets to the TG from other NIC. Test case count: 6.

Total 25 NSH_SFC functional tests in the CSIT-18.07.

6.2 Release Notes

6.2.1 Changes in CSIT-18.07

No changes.

6.2.2 Known Issues

No known issues.

6.3 Test Environment

FD.io CSIT NSH_SFC functional tests are executed in the same FD.io VIRT virtual environment as used CSIT VPP functional tests. See description in [VPP Functional Tests Environment](#).

6.4 Documentation

[CSIT NSH_SFC Functional Tests Documentation](#)¹⁶³ contains detailed functional description and input parameters for each test case.

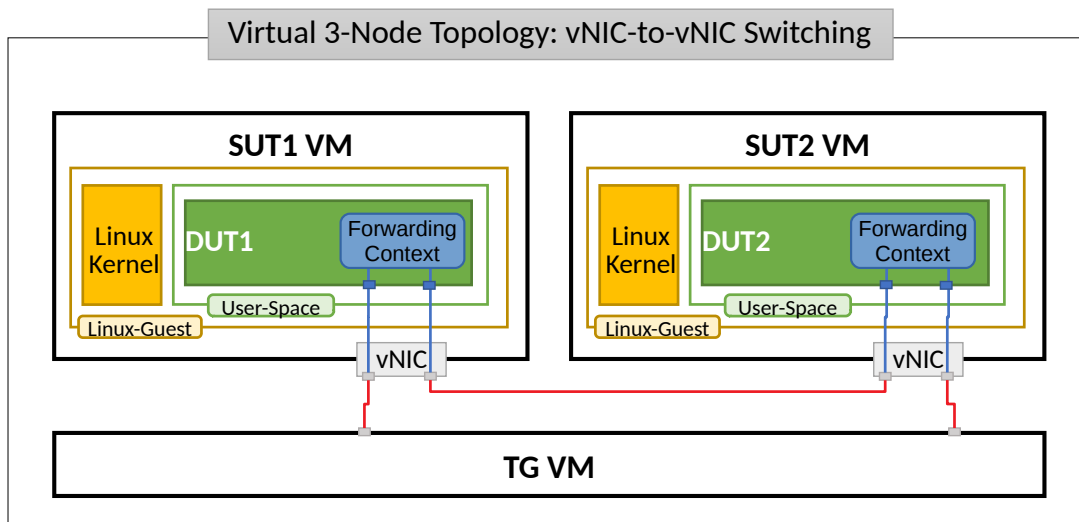
¹⁶³ https://docs.fd.io/csit/rls1807/doc/tests.nsh_sfc.func.html

7.1 Overview

7.1.1 Virtual Topologies

CSIT DMM functional tests are executed in VM-based virtual topologies created on demand using VIRT simulation platform contributed by Cisco. VIRT runs on physical baremetal servers hosted by LF FD.io project.

All tests are executed in three-node virtual test topology shown in the figure below.



SUT1 and SUT2 are two VMs (running Ubuntu), TG is a Traffic Generator VM (running Ubuntu). SUTs run sample server-client application using the DMM libraries in Linux user- mode as a Device Under Test (DUT) within the VM. Currently TG node is not being used in DMM-CSIT. Network connectivity between SUTs and to TG is provided using virtual NICs and VMs' virtio drivers.

7.1.2 Functional Tests Coverage

CSIT-18.07 includes following DMM functionality tested in virtual VM environment:

Function-ality	Description
DMM ba-sic opera-tion	DMM has only one test case right now. The testcase demonstrates single server[DUT1] and single client[DUT2] scenario using DMM framework and kernel tcp/ip stack. Test case count: 1.

7.2 Release Notes

7.2.1 Changes in CSIT-18.07

1. DMM FUNCTIONAL TESTS

- Added DMM base vs. epoll test case.

7.2.2 Known Issues

Here is the list of known issues in CSIT-18.07 for DMM functional tests in VIRT:

#	JiraID	Issue Description
1	DMM-17¹⁶⁴	DMM CentOS job disabled. DPDK version and relative kernel version issue, CentOS job is temporarily disabled.

7.3 Test Environment

FD.io CSIT DMM functional tests are executed in the same FD.io VIRT virtual environment as used CSIT VPP functional tests. See description in [VPP Functional Tests Environment](#).

7.4 Documentation

[CSIT DMM Functional Tests Documentation¹⁶⁵](#) contains detailed functional description and input parameters for each test case.

¹⁶⁴ <https://jira.fd.io/browse/DMM-17>

¹⁶⁵ <https://docs.fd.io/csit/rls1807/doc/tests.dmm.func.html>

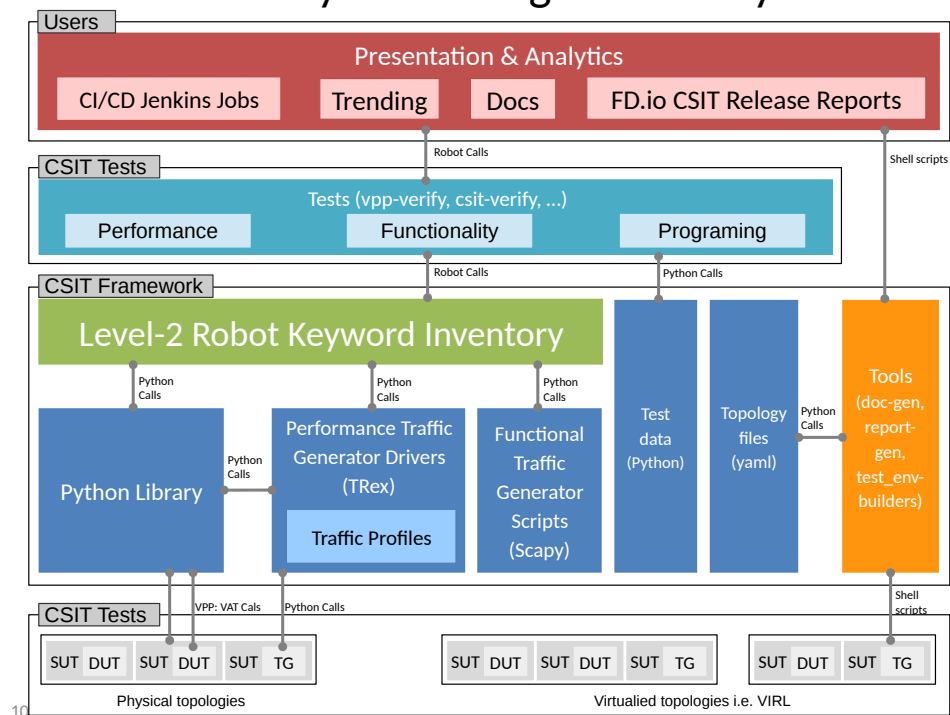
8.1 Design

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.

8.1.1 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.

CSIT System Design Hierarchy



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;
 - Functional tests using VIRT environment:
 - VPP;
 - Honeycomb;
 - NSH_SFC;
 - Performance tests using physical testbed environment:
 - VPP;
 - DPDK-Testpmd;
 - DPDK-L3Fwd;
 - Honeycomb;
 - VPP Container K8s orchestrated topologies;
 - Tools:
 - Documentation generator;
 - Report generator;
 - Testbed environment setup ansible playbooks;
 - Operational debugging scripts;

8.1.2 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 common 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;

8.1.3 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;

8.1.4 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.

8.1.5 Coding guidelines

Coding guidelines can be found on [Design optimizations wiki page](#)¹⁶⁶.

¹⁶⁶ https://wiki.fd.io/view/CSIT/Design_Optimizations

8.2 Test Naming

8.2.1 Background

CSIT-18.07 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](#)¹⁶⁷. Below few illustrative examples of the naming usage for test suites across CSIT performance, functional and Honeycomb management test areas.

8.2.2 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.

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.

8.2.3 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.

¹⁶⁷ <https://wiki.fd.io/view/CSIT/csit-test-naming>

- *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](#)¹⁶⁸.

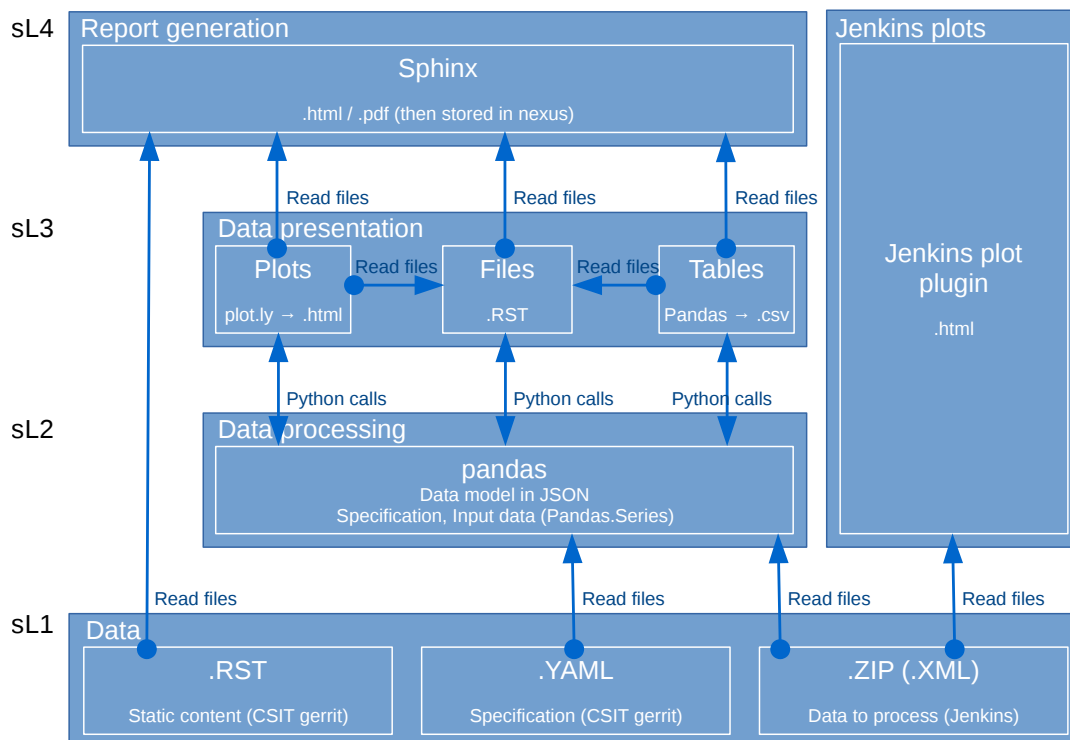
¹⁶⁸ <https://wiki.fd.io/view/CSIT/csit-test-naming>

8.3 Presentation and Analytics Layer

8.3.1 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`¹⁶⁹.
 - 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)



¹⁶⁹ <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html>

8.3.2 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"  
-  
  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]: "${DIR[WORKING]}/data"
  ## Static source files from git
  DIR[WORKING,SRC]: "${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]: "${DIR[DTR]}/vpp_performance_results"
  DIR[DTR,PERF,HC]: "${DIR[DTR]}/honeycomb_performance_results"
  DIR[DTR,FUNC,VPP]: "${DIR[DTR]}/vpp_functional_results"
  DIR[DTR,FUNC,HC]: "${DIR[DTR]}/honeycomb_functional_results"
  DIR[DTR,FUNC,NSHSFC]: "${DIR[DTR]}/nshsfc_functional_results"
  DIR[DTR,PERF,VPP,IMPRV]: "${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"

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

- 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"
  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|l2bdbasemac|rn)-ndrdisc"
  algorithm: "plot_performance_box"
  output-file-type: ".html"
  output-file: "{DIR[STATIC,VPP]}/64B-1t1c-l2-se11-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:

```

(continues on next page)

(continued from previous page)

```

- "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"
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-nsh_sfc-verify-func-1707-ubuntu1604-v1r1:
  -
    build: 2
    file: "csit-nsh_sfc-verify-func-1707-ubuntu1604-v1r1-2.xml"
  csit-vpp-functional-1707-ubuntu1604-v1r1:
  -
    build: lastSuccessfulBuild
    file: "csit-vpp-functional-1707-ubuntu1604-v1r1-lastSuccessfulBuild.xml"
  hc2vpp-csit-integration-1707-ubuntu1604:
  -
    build: lastSuccessfulBuild
    file: "hc2vpp-csit-integration-1707-ubuntu1604-lastSuccessfulBuild.xml"
  csit-vpp-perf-1707-all:
  -

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```
- 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-vir1:
- lastSuccessfulBuild
hc2vpp-csit-perf-master-ubuntu1604:
- 8
- 9
hc2vpp-csit-integration-1707-ubuntu1604:
- lastSuccessfulBuild
csit-nsh_sfc-verify-func-1707-ubuntu1604-vir1:
- 2
```

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: "${DIR[DTR,PERF,VPP,IMPRV]}/tpl_performance_improvements.csv"
output-file-ext: ".csv"
output-file: "${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]"

```

(continues on next page)

(continued from previous page)

```

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"
  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|l2bdbasemac|rn)-ndrdisc"
algorithm: "plot_performance_box"
output-file-type: ".html"
output-file: "{DIR[STATIC,VPP]}/64B-1t1c-l2-se11-ndrdisc"
data:
  csit-vpp-perf-1707-all:
    - 9
    - 10
    - 13

```

(continues on next page)

(continued from previous page)

```

- 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"
  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
    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

```

(continues on next page)

(continued from previous page)

```

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-http-server-performance” set and plot layout “plot-cps”):

```

-
type: "plot"
title: "VPP HTTP Server Performance"
algorithm: "plot_http_server_performance_box"
output-file-type: ".html"
output-file: "{DIR[STATIC,VPP]}/http-server-performance-cps"
data:
  "plot-vpp-http-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.. |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.

8.3.3 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

```

(continues on next page)

(continued from previous page)

```

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": {
    "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": {

```

(continues on next page)

(continued from previous page)

```

    "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",
    "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
    ...
  - 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:

$$N_core_throughput_speedup = \frac{N_core_throughput}{1_core_throughput}$$

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 repeatability 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_performance_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"
  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.

8.3.4 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.ly¹⁷⁰ 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.

8.3.5 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).

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.

¹⁷⁰ <https://plot.ly/>

8.3.6 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:
 - (a) By PT job at its completion.
 - (b) Manually from Jenkins UI.
2. Download and parse archived historical data and the new data:
 - (a) New data from latest PT job is evaluated against the rolling window of <N> sets of historical data.
 - (b) Download RF output.xml files and compressed archived data.
 - (c) 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:
 - (a) Calculate quartiles Q1, Q2, Q3.
 - (b) Trim outliers using IQR.
 - (c) Calculate TMA and TMSD.
 - (d) Calculate normal trending range per test case based on TMA and TMSD.
4. Evaluate new test data against trend metrics:
 - (a) If within the range of $(TMA \pm 3 \cdot TMSD)$ => Result = Pass, Reason = Normal.
 - (b) If below the range => Result = Fail, Reason = Regression.
 - (c) If above the range => Result = Pass, Reason = Progression.
5. Generate and publish results
 - (a) Relay evaluation result to job result.
 - (b) Generate a new set of trend analysis summary graphs and drill-down graphs.
 - i. Summary graphs to include measured values with Normal, Progression and Regression markers. MM shown in the background if possible.
 - ii. Drill-down graphs to include MM, TMA and TMSD.
 - (c) 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 'L2XC fwd') and not 'VHOST' and not 'MEMIF'"
    parameters:
      - "result"
    layout: "plot-cpta-vpp"
  - title: "DPDK 4T4C IMIX MRR Trending"
    output-file-name: "dppk-imix-4t4c-xl710"
    data: "plot-performance-trending-dppk"
    filter: "'NIC_Intel-XL710' and 'IMIX' and 'MRR' and '4T4C' and 'DPDK'"
    parameters:
      - "result"
    layout: "plot-cpta-dppk"

```

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_performance_trending_dashboard"
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

```

(continues on next page)

(continued from previous page)

```
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_performance_trending_dashboard_html"
input-file: "{DIR[STATIC,VPP]}/performance-trending-dashboard-1t1c.csv"
output-file: "{DIR[STATIC,VPP]}/performance-trending-dashboard-1t1c.rst"
```

8.3.7 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:
 - (a) Summary graphs to include measured values with Progression and Regression markers.
 - (b) 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

8.3.8 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:

- data
- metadata
- suites
- tests

environment.py

Functions:

- clean_environment

class Environment

Methods:

- set_environment

Getters:

- environment

input_data_files.py

(continues on next page)

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

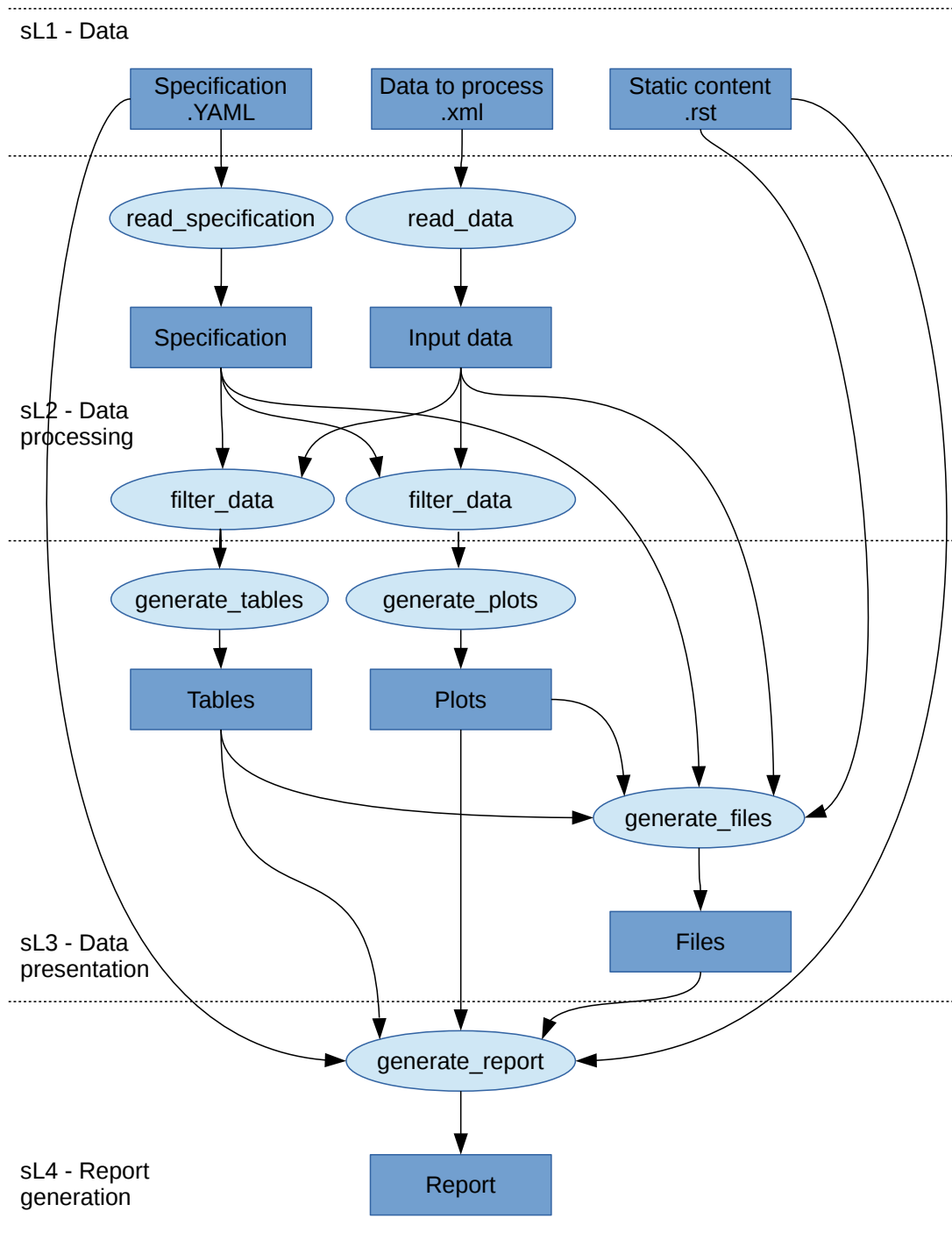
report.py

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.

8.4 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](#)¹⁷¹.

8.4.1 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.

8.4.2 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).

8.4.3 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.

¹⁷¹ https://git.fd.io/csit/tree/docs/tag_documentation.rst?h=rls1807

8.4.4 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.

8.4.5 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

TNL_1000

IPSec in tunnel mode - 1000 tunnels.

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.

8.4.6 Tags marking functional vs. performance of tests

FUNCTEST

All functional test cases.

PERFTEST

All performance test cases.

8.4.7 Performance testing 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.

MRR

Performance tests where TG sends the traffic at maximum rate (line rate) and reports total sent/received packets over performance trial duration.

8.4.8 Ethernet frame size tags for performance tests

64B

64B frames used for test.

78B

78B frames used for test.

114B

114B frames used for test.

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.

8.4.9 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.

8.4.10 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.

8.4.11 Underlay tags**IP4UNRLAY**

IPv4 underlay.

IP6UNRLAY

IPv6 underlay.

MPLSUNRLAY

MPLS underlay.

8.4.12 Overlay tags**L2OVLAY**

L2 overlay.

IP4OVLAY

IPv4 overlay (IPv4 payload).

IP6OVLAY

IPv6 overlay (IPv6 payload).

8.4.13 Tagging tags**DOT1Q**

All test cases with dot1q.

DOT1AD

All test cases with dot1ad.

8.4.14 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.

8.4.15 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_DPK

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.

8.4.16 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.

8.4.17 Encryption tags

IPSECSW

Crypto in software.

IPSECHW

Crypto in hardware.

IPSECTRAN

IPSec in transport mode.

IPSECTUN

IPSec in tunnel mode.

8.4.18 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.

DOCKER

All test cases which use Docker as container manager.

APP

All test cases with specific APP use.

8.4.19 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.

8.4.20 Multi-threading tags

STHREAD

Dynamic tag. All test cases using single poll mode thread.

MTHREAD

Dynamic tag. All test cases using more than 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.

8.4.21 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](#)⁹¹
- [lxcnamespace] [Resource management: Linux kernel Namespaces and cgroups](#)⁹².
- [stgraber] [LXC 1.0: Blog post series](#)⁹³.
- [lxcsecurity] [Linux Containers Security](#)⁹⁴.
- [capabilities] [Linux manual - capabilities - overview of Linux capabilities](#)⁹⁵.
- [cgroup1] [Linux kernel documentation: cgroups](#)⁹⁶.
- [cgroup2] [Linux kernel documentation: Control Group v2](#)⁹⁷.
- [selinux] [SELinux Project Wiki](#)⁹⁸.
- [lxcsecfeatures] [LXC 1.0: Security features](#)⁹⁹.
- [lxcsource] [Linux Containers source](#)¹⁰⁰.
- [apparmor] [Ubuntu AppArmor](#)¹⁰¹.
- [seccomp] [SECure COMPUting with filters](#)¹⁰².
- [docker] [Docker](#)¹⁰³.
- [k8sdoc] [Kubernetes documentation](#)¹⁰⁴.
- [ligato] [Ligato](#)¹⁰⁵.
- [govpp] [FD.io goVPP project](#)¹⁰⁶.
- [vppagent] [Ligato vpp-agent](#)¹⁰⁷.

⁹¹ <https://linuxcontainers.org/>

⁹² <https://www.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/lxc-namespace.pdf>

⁹³ <https://stgraber.org/2013/12/20/lxc-1-0-blog-post-series/>

⁹⁴ <https://linuxcontainers.org/lxc/security/>

⁹⁵ <http://man7.org/linux/man-pages/man7/capabilities.7.html>

⁹⁶ <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>

⁹⁷ <https://www.kernel.org/doc/Documentation/cgroup-v2.txt>

⁹⁸ http://selinuxproject.org/page/Main_Page

⁹⁹ <https://stgraber.org/2014/01/01/lxc-1-0-security-features/>

¹⁰⁰ <https://github.com/lxc/lxc>

¹⁰¹ <https://wiki.ubuntu.com/AppArmor>

¹⁰² https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt

¹⁰³ <https://www.docker.com/what-docker>

¹⁰⁴ <https://kubernetes.io/docs/home/>

¹⁰⁵ <https://github.com/ligato>

¹⁰⁶ <https://wiki.fd.io/view/GoVPP>

¹⁰⁷ <https://github.com/ligato/vpp-agent>

[imagevar] Image parameter is required in initial commit version. There is plan to implement container build class to build Docker/LXC image.