High Performance Cloud-native Networking
K8s Unleashing FD.io

Maciek Konstantynowicz et al in FD.io CSIT
FD.io CSIT Project Lead, If-id/iirc: mackonstan
Distinguished Engineer, Cisco, mkonstan@cisco.com

10-Dec-2018 FD.io mini-summit
Kubecon Cloud-Native-Con North America, Seattle
DISCLAIMERS

• 'Mileage May Vary'
  • Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your opinion and investment of any resources. For more complete information about open source performance and benchmark results referred in this material, visit https://wiki.fd.io/view/CSIT and/or https://docs.fd.io/csit/rls1810/report/.

• Trademarks and Branding
  • This is an open-source material. Commercial names and brands may be claimed as the property of others.
Internet Mega Trends – ..

Portability and Efficiency

Scalability and Self-Healing

Software Defined Networking

Cloud Native Designs

Open Source Platforms

Cloud

NFV

SDN
Remember **1965** "Moore’s Law" –..

Remember **1965** "Moore’s Law" –..

Remember 1965 "Moore’s Law" – Does It Still Work?

Remember 1965 "Moore’s Law" – Does It Still Work?

Remember **1965** "Moore’s Law" – Yes, It Surely Does ..

“Ramble On ..”

Processing Packs: How to Use Compute ...

Resources to Get Performance

1. **Processor and CPU cores**
   *for performing* packet processing operations

2. **Memory bandwidth**
   *for moving* data (packets, lookup) and instructions (packet processing)

3. **I/O bandwidth**
   *for moving* packets to/from NIC interfaces

4. **Inter-socket bandwidth**
   *for handling* inter-socket operations

---

**CyclesPerPacket {ClockCycles} = \frac{\#Instructions}{Packet} \times \#Cycles_{instruction}**

**Throughput [pps] = \frac{1}{Packet_{Processing\ Time}[sec]} = \frac{CPU\_freq[Hz]}{Cycles\_per\_Packet}**

**Throughput [bps] = Throughput[pps] \times Packet\_Size[pps]**
Processing Packets: What Improves in Compute ..

Generation to Generation – BDX to SKX example

Resources to Get Performance

1. Processor and CPU cores
   - **FrontEnd:** faster instr. decoder (4- to 5-wide)
   - **BackEnd:** faster L1 cache, bigger L2 cache, deeper OOO* execution
   - **Uncore:** move from ring to X-Y fabric mesh

2. Memory bandwidth
   - ~50% increase: channels (4 to 6), speed (DDR-2666)

3. I/O bandwidth
   - >50% increase: PCIe lanes (40 to 48), re-designed IO blocks

4. Inter-socket bandwidth
   - ~60% increase: QPI to UPI (2x to 3x), interface speed (9.6 to 10.4 GigTrans/sec)

Moore’s Law in Action

\[
\text{Cycles Per Packet} = \frac{\text{#Instructions}}{\text{Packet Size}} \times \frac{\text{Cycles Per Instruction}}{\text{Packet Size}}
\]

\[
\text{Throughput} = \text{Throughput [bps]} \times \text{Packet Size [pps]}
\]
Open Source Benchmarking – Guiding Principles

- Discover the **limits** and **know** them
- Assess based on **externally measured data** and behaviour (black-box)
- Guide benchmarking by **good understanding** of the whole system (white-box)
- Provide a **feedback loop** to hardware and software engineering

“One should understand the laws of physics, and use them efficiently!”
“Without data, you're just another person with an opinion.” — W. Edwards Deming
FD.io CSIT-1810 Open-Source Benchmarking Report

VPP Performance:

1. VPP release notes

2. VPP performance graphs

3. VPP performance comparisons
   c. 2-Node Skylake vs. 3-Node Skylake testbeds: [http://bit.ly/2FzC84q](http://bit.ly/2FzC84q)

4. VPP performance test details

DPDK Testpmd and L3fwd Performance:

1. DPDK release notes

2. DPDK Testpmd and L3fwd performance graphs

3. DPDK performance comparisons

FD.io CSIT-1810.49 report has been published on FD.io docs site:

- html: https://docs.fd.io/csit/rls1810/report/
FD.io CSIT-1810 Open-Source Benchmarking Report

Performance Test Environments: Physical Testbeds

SUT servers are populated with:
- NIC-1: x710-DA4 4p10GE Intel.
- NIC-3: mcx556a-edat ConnectX5 2p100GE Mellanox. (Not used yet.)

SUT1 and SUT2 servers are populated:
- NIC-1: x710-DA4 4p10GE Intel.
FD.io CSIT-1810 Open-Source Benchmarking Report
Performance Test Environments: Logical Topologies
Benchmarks Data and Public References

- Multi-Platform/Vendor
  - Intel Xeon and Atom, Arm (WIP)
- Packet Throughput & Latency
  - Zero-Drop & Partial Drop Rates
- Data Plane Workloads
  - FD.io VPP
  - DPDK L3fwd, Testpmd
- Scaling
  - Single-, Multi-Core
  - MACs, IPs, Flows, ACLs etc.
  - K8s
- Performance Test Suites (unique #)
  - L2: 106
  - VM vhostuser: 56
  - Containers memif: 12
  - Crypto: 26
  - SRv6: 12
  - TCP/IP: 02
  - Total: 350

FD.io CSIT
Per release test and performance reports

Breath (# of test cases), Depth (of measurement) and Repeatability (every release, repeatable locally)

CSIT-1810 report includes Ever More benchmarking data, more graphs (756 on the last count) that are now more orderly and more beautifully presented than Ever. Report HTML and PDF layouts got revamped more than a bit, striving to improve presentation clarity and browsability. And equally portability and consumability got improved. We are aiming the FD.io CSIT Open-Source Benchmarking reports to be consumable by everyone. And by everyone we mean Everyone. Those who code it, those who test it, those who use it, those who want to use it but are afraid to touch it. Everyone who likes it.

https://docs.fd.io/csit/rls1810/report/
FD.io CSIT-1810: Packet Throughput Results

Source: https://docs.fd.io/csit/rls1810/report/
FD.io CSIT-1810: Throughput Speedup Results

FD.io VPP Multi-Core Speedup Properties:

A. Predictable performance
B. Linear scaling with cores
C. Follows Amdahl’s Law
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10ge2p1x710-64b-1c-ethip4-ip4scale2m</td>
<td>8.51</td>
<td>0.0</td>
<td>15.47</td>
<td>0.02</td>
<td>81</td>
</tr>
<tr>
<td>10ge2p1x710-78b-1c-ethip4-ip4scale2m</td>
<td>4.99</td>
<td>0.0</td>
<td>8.62</td>
<td>0.05</td>
<td>72</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-ethip4udp-ip4base-iac150sf-10kflows</td>
<td>5.85</td>
<td>0.0</td>
<td>9.78</td>
<td>0.02</td>
<td>67</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-eth-l2dbuscale1maclrn</td>
<td>13.56</td>
<td>0.0</td>
<td>22.7</td>
<td>0.07</td>
<td>67</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-ethip4udp-ip4base-iac150sf-10kflows</td>
<td>5.85</td>
<td>0.0</td>
<td>9.8</td>
<td>0.05</td>
<td>67</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn</td>
<td>6.92</td>
<td>0.0</td>
<td>11.38</td>
<td>0.04</td>
<td>64</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>5.87</td>
<td>0.0</td>
<td>8.34</td>
<td>0.02</td>
<td>64</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>6.61</td>
<td>0.0</td>
<td>18.78</td>
<td>0.03</td>
<td>63</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2xbase</td>
<td>15.99</td>
<td>0.0</td>
<td>26.19</td>
<td>0.01</td>
<td>63</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-ethip4udp-ip4base-iac150sf-10kflows</td>
<td>11.73</td>
<td>0.0</td>
<td>19.2</td>
<td>0.07</td>
<td>63</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-ethip4udp-ip4base-iac150sf-10kflows</td>
<td>11.77</td>
<td>0.0</td>
<td>19.17</td>
<td>0.1</td>
<td>62</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2xbase-eth-1memif-1dcr</td>
<td>10.85</td>
<td>0.0</td>
<td>17.36</td>
<td>0.02</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>10.13</td>
<td>0.0</td>
<td>16.25</td>
<td>0.09</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>7.8</td>
<td>0.0</td>
<td>12.48</td>
<td>0.03</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>10.12</td>
<td>0.0</td>
<td>16.25</td>
<td>0.26</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>10.12</td>
<td>0.0</td>
<td>16.25</td>
<td>0.26</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>7.52</td>
<td>0.0</td>
<td>12.1</td>
<td>0.03</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>15.85</td>
<td>0.0</td>
<td>24.1</td>
<td>0.1</td>
<td>60</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>6.53</td>
<td>0.0</td>
<td>18.51</td>
<td>0.06</td>
<td>59</td>
</tr>
<tr>
<td>10ge2p1x710-64b-2c-ethip4udp-ip4scale1000-udpsrcscale15-nat44</td>
<td>12.85</td>
<td>0.0</td>
<td>20.46</td>
<td>0.09</td>
<td>59</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>11.22</td>
<td>0.0</td>
<td>17.80</td>
<td>0.02</td>
<td>59</td>
</tr>
<tr>
<td>10ge2p1x710-78b-2c-ethip4udp-ip4scale20k</td>
<td>15.67</td>
<td>0.0</td>
<td>24.76</td>
<td>0.04</td>
<td>58</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-dotlq-l2xbase</td>
<td>18.54</td>
<td>0.0</td>
<td>16.44</td>
<td>0.02</td>
<td>55</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-ethip4udp-ip4base-iac150sf-10kflows</td>
<td>5.76</td>
<td>0.0</td>
<td>8.95</td>
<td>0.09</td>
<td>55</td>
</tr>
<tr>
<td>10ge2p1x710-64b-1c-eth-l2dbuscale1maclrn-iac150sf-10kflows</td>
<td>12.4</td>
<td>0.0</td>
<td>19.34</td>
<td>0.02</td>
<td>55</td>
</tr>
</tbody>
</table>

https://docs.fd.io/csit/rls1810/report/_static/vpp/performance-compare-testbeds-3n-hsw-3n-skx-mrr.txt
FD.io CSIT-1901: NFV Service Density Measurements
Work resulting from collaboration with CNCF CI team

Problem

Clear performance metrics and standards are missing, despite SDN and NFV services being on the rise for few years now, and with many questions unanswered: how many (qty) and how much ($) NFV services can be supported on compute node(s) ?

How much compute resources is consumed (e.g. cores), how do VM and Container technologies compare, VNF and CNF ?

Applicability

Applies to all user verticals either already operating or moving to SDN NFV network designs.
Developers, CD/CI Integrators, Network Engineers, Designers, Decision Makers.

Solution

Benchmark NFV Service Densities using Virtual Machine and Container technologies, single compute node measurements for now.
Leveraged a collective cross-team experience of FD.io CSIT and CNCF dev and test teams.

WARNING WHAT FOLLOWS IS VERY FRESH OFF THE PRESS
PLEASE PROCEED WITH CAUTION !

IT IS FOCUSING ON TECHNOLOGY BENCHMARKING AND COMPARISON
NOT A PRODUCT – TREAT IT AS SOFT LAUNCH ;(}
NFV Technologies: VNF and CNF

**Virtualized Network Function**
- Virtual machine based
- Larger compute footprint
- Slower to start, stop, change
- Hard to move around

**Cloudified Network Function**
- Linux container based
- Smaller compute footprint
- Faster to start, stop, change
- Easy to move around

**Q1**: How to compare NF technologies like VNF and CNF?
**A**: Standalone not doing any work. Eh?
**A**: Inter-connected doing packet processing work. Yes!

**Q2**: How to measure NF compute footprint?
**A**: Processing packets. Yes!
**A**: Forwarding packets over inter-connects. Yes!

**Q3**: How to benchmark NFV services based on any NF technologies and any HW and SW stack?
**A**: Inter-connect few of them *somehow* into *some* useful topology, and measure some work. Eh?
**A**: Define an NF inter-connection model and benchmark NF service density using it. Yes!
**NFV Technologies: VNF and CNF**

<table>
<thead>
<tr>
<th>Virtualized Network Function</th>
<th>Cloudified Network Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machine based</td>
<td>Linux container based</td>
</tr>
<tr>
<td>Larger compute footprint</td>
<td>Smaller compute footprint</td>
</tr>
<tr>
<td>Slower to start, stop, change</td>
<td>Faster to start, stop, change</td>
</tr>
<tr>
<td>Hard to move around</td>
<td>Easy to move around</td>
</tr>
</tbody>
</table>

**Q1**: How to compare NF technologies like VNF and CNF?
- **A**: Standalone not doing any work. *Eh?*
- **A**: Inter-connected doing packet processing work. *Yes!*

**Q2**: How to measure NF compute footprint?
- **A**: Processing packets. *Yes!*
- **A**: Forwarding packets over inter-connects. *Yes!*

**Q3**: How to benchmark NFV services based on any NF technologies and any HW and SW stack?
- **A**: Inter-connect few of them *somehow* into some useful topology, and measure some work. *Eh?*
- **A**: Define an NF inter-connection model and benchmark NF service density using it. *Yes!*

**NFV Service Density Matrix** is a particular methodology of addressing Q1, Q2 and Q3. It is a result of collaboration across open source projects: **FD.io CSIT, CNCF/CNFs** and **OPNFV/NFVbench**.
NFV Service Design – with VNFs and CNFs

Single Compute Node

NFV Node

NFV Node with VNFs

NFV Node with CNFs

* Linux User-Mode SW Switch, Linux Kernel-Mode Networking, Direct Virtual Function, ..
**NFV Service Abstraction**

**Service Forwarding Graph**
How are the network functions and network devices forwarding packets

**Service Topology**
How are the network functions and network devices configured

**Network Topology**
How are the network functions and network devices interconnected
Service Topologies, Defined and Testing So Far..

**Service Topology**

**Forwarding Graph**

**VNF Service Chain (VSC)**

Snake Forwarding

**CNF Service Chain (CSC)**

Snake Forwarding

**CNF Service Pipeline (CSP)**

Pipeline Forwarding
Service Density Matrix: Network Function View

More is Better

Row:  **1..10** Network Service Instances.
Column:  **1..10** Network Functions per Service Instance.
Value:  **1..100** Network Functions.

<table>
<thead>
<tr>
<th>SVC</th>
<th>001</th>
<th>002</th>
<th>004</th>
<th>006</th>
<th>008</th>
<th>010</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>006</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>008</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>64</td>
<td>80</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>
Per NF Physical Core Resource Allocation

 Allocation of processor physical cores per NF instance is as follows:

1. **Two mapping ratios are defined and used in NF service matrix benchmarking:**
   a. `pcdr4nf` value determines Physical Core to Dataplane Ratio **for NF**.
   b. `pcmr4nf` value determines Physical Core to Main Ratio **for NF**.

2. **Target values to be benchmarked:**
   a. `pcdr4nf` = [(1:1),(1:2),(1:4)].
   b. `pcmr4nf` = [(1:2),(1:4),(1:8)].

3. **Number of physical cores required for the benchmarked NFs' service matrix is calculated as follows:**
   - \( \#pc = pcdr4nf \times \#dnf + pcmr4nf \times \#mnf \)
   - where
   - \( \#pc \) – total number of physical cores required and used.
   - \( \#dnf \) – total number of NF dataplane thread sets (1 set per NF instance).
   - \( \#mnf \) – total number of NF main thread sets (1 set per NF instance).

Source: https://github.com/cncf/cnfs/blob/master/comparison/doc/cncf-cnfs-results-summary.md
Service Density Matrix: Core Usage View

Less is Better!

Row: 1..10 Network Service Instances.
Column: 1..10 Network Functions per Service Instance.
Value: 1..NN Processor Physical Cores Used.

pcdr4nf = (1:1), pcmr4nf = (1:2)

<table>
<thead>
<tr>
<th>SVC</th>
<th>001</th>
<th>002</th>
<th>004</th>
<th>006</th>
<th>008</th>
<th>010</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>002</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>004</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>006</td>
<td>9</td>
<td>18</td>
<td>36</td>
<td>54</td>
<td>72</td>
<td>90</td>
</tr>
<tr>
<td>008</td>
<td>12</td>
<td>24</td>
<td>48</td>
<td>72</td>
<td>96</td>
<td>120</td>
</tr>
<tr>
<td>010</td>
<td>15</td>
<td>30</td>
<td>60</td>
<td>90</td>
<td>120</td>
<td>150</td>
</tr>
</tbody>
</table>

Source: https://github.com/cncf/cnfs/blob/master/comparison/doc/cncf-cnfs-results-summary.md
Service Density Matrix: MRR Throughput Results
More is Better!

VNF Service Chains (VSC)

CNF Service Chains (CSC)

CNF Service Pipelines (CSP)

Source: https://github.com/cncf/cnfs/blob/master/comparison/doc/cncf-cnfs-results-summary.md
Service Density Matrix: What’s Next..

Work in progress:

• Service Density Matrix in FD.io CSIT-1901
  • VSC, CSC, CSP [1..10] x [1..10] – continuous tests MRR, MLRsearch, PLRsearch
  • Xeon, Arm, Atom Platforms

• Continue collaboration with NSM team
• Continue collaboration with CNCF CI team

In work queue:

• Once “ready for service” add Network Service Mesh orchestration
• Add promising new virtualization and containerization technologies
  • “Micro” VMs: AWS Firecracker, https://firecracker-microvm.github.io/
  • More secure and faster containers: Singularity Containers, https://github.com/sylabs/singularity

AFTER A SOFT LAUNCH
COMES THE HARD LAUNCH ;)

networkservicemesh.io
Benchmarking standardization: IETF, ETSI

Done or work in progress:

- Two IETF drafts in bmwg
  - Aiming to update RFC2544, RFC1242
  - markdown source files in FD.io CSIT repo, [https://git.fd.io/csit/tree/docs/ietf](https://git.fd.io/csit/tree/docs/ietf)
- ETSI NFV
  - NFV vswitch benchmarking
  - FD.io CSIT contributions to the last version of technical report ver. TST009_v0015

In work queue:

- One IETF draft for bmwg
  - Aiming to update RFC2544, RFC1242

* Link not active, .md document will show up first in git.fd.io/csit/tree/docs/ietf.
Packet Vectors are Good for You!

**Netgate** shipping product(s) [1]

**Alibaba** [2]

---


5 Pillars of Modern Software Data Planes

Blazingly Fast
- Process the *massive explosion* of East-West traffic
- Process *increasing* North-South traffic

Truly Extensible
- Foster *pace of innovation* in cloud-native networking
- *No compromise* on performance (zero-tolerance)

Software First
- Cloud means *running everywhere*
- Cloud means hardware and physical *infra agnostic*

Predictable Performance
- Dataplane performance must be deterministic
- Predictable for a number of VMs, Containers, virtual topology and (E-W, N-S) traffic matrix

Measureable
- Counters everywhere to *count everything* for detailed cross-layer operation and efficiency monitoring
- Enables feedback loop to drive engineering optimizations

FD.io VPP meets these challenges
Baremetal Data Plane Performance Limit
FD.io benefits from increased Processor I/O

YESTERDAY

Intel® Xeon® E5-2699v4
22 Cores, 2.2 GHz, 55MB Cache

- Network I/O: 160 Gbps
- Core ALU: 4-wide parallel µops
- Memory: 4-channels 2400 MHz
- Max power: 145W (TDP)

TODAY

Intel® Xeon® Platinum 8168
24 Cores, 2.7 GHz, 33MB Cache

- Network I/O: 280 Gbps
- Core ALU: 5-wide parallel µops
- Memory: 6-channels 2666 MHz
- Max power: 205W (TDP)

FD.io Takes Full Advantage of Faster Intel® Xeon® Scalable Processors
No Code Change Required

https://goo.gl/UtbaHy

Breaking the Barrier of Software Defined Network Services
1 Terabit Services on a Single Intel® Xeon® Server!
Internet Mega Trends – ..

- Portability and Efficiency
- Scalability and Self-Healing
- Software Defined Networking
- Cloud Native Designs
- Open Source Platforms

Cloud
NFV
SDN
Internet Mega Trends – *Being* Addressed ..
Internet Mega Trends – *Being Addressed* ..

**Portability and Efficiency**
Public, private, hybrid, any-cloud. Over 10 times faster Container networking vs. alternatives.

**Scalability and Self-healing**
Follows Kubernetes scale and self-healing principles.

**Software Defined Networking**
FD.io VPP, the Fastest SW Data Plane on the Planet. Over 200 programmable “micro-NFs” and plugins.

**Cloud Native Designs**
Containerized NFs managed as true cloud-native apps, provide and consume dat plane microservices.

**Open Source Platforms**
Based on the best-of-breed collaborative projects in Linux Foundation.
References

FD.io VPP, CSIT and related projects
- VPP: https://wiki.fd.io/view/VPP
- CSIT-CPL: https://wiki.fd.io/view/CSIT
- pma_tools - https://wiki.fd.io/view/Pma_tools

Benchmarking Methodology

NFV Service Density
CNCF CNF Project Links

- Collaboration with FD.io CSIT and OPNFV-NFVbench
  - FD.io CSIT
  - OPNFV-NFVbench
- Repo: https://github.com/cncf/cnfs
  - Comparison test case summary and overview
- Comparison and code examples:
  - CNF Edge Throughput and Baseline Single Chain
    - CNF and VNF
    - VPP vSwitch
  - Baseline NF Performance Test
  - In progress: Baseline K8s Chained NF Test
  - Github Projects and Issues
FD.io CSIT Project Links

We invite you to Participate in FD.io

- Get the Code, Build the Code, Run the Code
- Try the vpp user demo
- Install vpp from binary packages (yum/apt)
- Read/Watch the Tutorials
- Join the Mailing Lists
- Join the IRC Channels
- Explore the wiki
- Join FD.io as a member

Thank you!
High Performance Cloud-native Networking
K8s Unleashing FD.io

THANK YOU !!
And Enjoy Seattle !