Accelerating the Development of Cloud-native C/VNFs

Giles Heron
Principal Engineer, Cisco
giheron@cisco.com

Maciek Konstantynowicz
FD.io CSIT Project Lead
Distinguished Engineer, Cisco
mkonstan@cisco.com

Damjan Marion
FD.io VPP Committer
Principal Engineer, Cisco
damarion@cisco.com
Agenda

• Context
• FD.io / VPP
• Ligato
• Memif
• The Numbers
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/rls1807/report/.

• Trademarks and Branding
  • This is an open-source material. Commercial names and brands may be claimed as the property of others.
SDN NFV Evolution to Cloud-native
Moving on from VMs to Pods/Containers

• Network function workloads moving from VMs to Containers
  Native code execution on compute nodes, much less execution overhead
  Lighter workloads, many more of them, much more dynamic environment

• Orchestration moving from OpenStack VMs to K8s Pods/Containers
  Pod/Container networking being addressed: Ligato, Network Services Mesh, Multus

• Pressing need for optimised user-mode packet virtual interface
  Equivalent of “virtio-vhostuser” for Containers, but much faster
  Must be compatible with Container orchestration stack
  Opportunity to do it right!

Should allow us to get closer to the native bare-metal limits ...
Bare-Metal Data Plane Performance Limit
FD.io benefits from increased Processor I/O

**YESTERDAY**
- **Server**
  - 2x [2 Sockets]
  - PCIe Packet Forwarding Rate [Gbps]: 640
  - +75%

**TODAY**
- **Server**
  - 2x [2 Sockets]
  - PCIe Packet Forwarding Rate [Gbps]: 1,120
  - +75%

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

**Network I/O**
- **Intel® Xeon® E5-2699v4**
  - 22 Cores, 2.2 GHz, 55MB Cache
  - **200 Gbps**

**Intel® Xeon® Platinum 8168**
- 24 Cores, 2.7 GHz, 33MB Cache
  - **280 Gbps**

**URL**
- [https://goo.gl/UtbaHy](https://goo.gl/UtbaHy)

**Breaking the Barrier of Software Defined Network Services**
1 Terabit Services on a Single Intel® Xeon® Server!
FD.io VPP – Vector Packet Processing
Compute-Optimised SW Networking Platform

Packet Processing Software Platform
• High performance
• Linux user space
• Runs on compute CPUs: and “knows” how to run them well!

Shipping at volume in server & embedded products
FD.io VPP – How does it work?
Compute Optimised SW Networking Platform

1. Packet processing is decomposed into a directed graph of nodes...
2. ... packets move through graph nodes in vector...
3. ... graph nodes are optimized to fit inside the instruction cache...

- Each graph node implements a “micro-NF”, a “micro-NetworkFunction” processing packets.

* Makes use of modern Intel® Xeon® Processor micro-architectures. Instruction cache & data cache always hot ➔ Minimized memory latency and usage.
Packet Processing

Packet Processing Graph

Input Graph Node

Graph Node

Packet

Vector of n packets

fd.io VPP - Architecture

Packet Processing

Packet Processing Graph

Vector of n packets

Input Graph Node

Graph Node

Packet

Graph Node

Packet Processing Graph

Vector of n packets

Input Graph Node

Graph Node

Packet

Packet Processing

Packet Processing Graph

Vector of n packets

Input Graph Node

Graph Node

Packet
fd.io VPP - Architecture
Splitting the Vector
**fd.io VPP - Architecture**

**Plugins**

- **Hardware Plugin**
  - hw-accel-input
  - SKIP S/W nodes where work is done by hardware already

- **Input Graph Node**
  - etherneth-input

- **Graph Node**
  - ip4-input
  - mpls-input
  - arp-input
  - ip6-input
  - ip4-lookup
  - ip6-lookup
  - ip6-reverse
  - ip6-local
  - ip4-local
  - ip4-reverse
  - custom-1
  - custom-2

- **Packet Processing Graph**
  - Packet
  - Vector of n packets
  - Input Graph Node
  - Graph Node
  - Plugin
    - /usr/lib/vpp_plugins/foo.so
    - First class citizens
    - That can:
      - Add graph nodes
      - Add API
      - Rearrange graph
    - Can be built independently of VPP source tree

**Plugins are:**

- First class citizens
  - That can:
    - Add graph nodes
    - Add API
    - Rearrange graph

**Can be built independently of VPP source tree**
Ligato CN-Infra: a CNF* Development Platform

www.github.com/ligato/cn-infra

* CNF – Cloud-native Network Function
Ligato VPP Agent: a CNF Management Agent
www.github.com/ligato/vpp-agent
Ligato Controller: a CNF Deployment Platform

www.github.com/ligato/sfc-controller

Core
Plugin
Lifecycle
Management

CN-Infra

API

Controller App

Lifecycle SPI

RPC

DB

Messaging

Logs

Health

App clients

KV Data Store (Etcd, Cassandra, Redis)

Kafka

Log Aggregator (e.g. Logstash)

Health Monitor (e.g. K8s)

CN-Infra Plugins

REST

Etcd

Redis

Cassandra

Kafka

Logrus

Log Manager

Probes

Status

Health Monitor

Log Aggregator (e.g. Logstash)

Kafka

KV Data Store (Etcd, Cassandra, Redis)

App clients

RPC
Ligato – Cloud-native Network Functions (CNF)
Putting It All Together Now – The Software Architecture

Functional Layered Diagram

- Service Policy
- Service Topology
- Lifecycle

Production-Grade Container Orchestration
- Kubernetes
- API Proxies

Network Function and Network Topology Orchestration
- SFC Controller
- Container Networking
- Contiv Netmaster

Containerized Network Data Plane
- Networking Plugin
- Contiv Netmaster
- Kubelet

Software Architecture Diagram

Control and Management Plane
- Applications
- SFC Controller
- Tools (e.g., agentd)

Message Bus
- etcd
- Configuration
- Operational State
- Kafka
- Configuration Notifications

Containerized Network Data Plane
- Containerized Network Functions
- Containerized Network Functions
- Containerized Network Functions

Containers Lifecycle Orchestration

Networking
- GoAGENT
- cSwitch
- CNF
- ...
Ligato – Cloud-native Network Functions (CNF)
Putting It All Together Now – The System Architecture

- Cloud-Native Networking (Kubernetes) is designed for applications, not NFV
- Ligato wires the NFV data plane together into a service topology
- Dedicated Telemetry Engine in VPP enables closed-loop control
- Offload functions to NIC but via vSwitch in host memory
Service Function Chaining with Ligato
Cloud-native Network Functions
Optimising Performance within the Compute Node

Getting closer to bare-metal speeds ...

With a New Cloud-native Network Packet Virtual Interface, memif
memif – Motivation

• Create packet based shared memory interface for user-mode application
• Be container friendly (no privileged containers needed)
• Support both polling and interrupt mode operation
  • Interrupts simulated with linux eventfd infrastructure
  • Support for interrupt masking in polling mode
• Support vpp-to-vpp, vpp-to-3rd-party and 3rd-party-to-3rd-party operation
• Support for multiple queues (incl. asymmetric configurations)
• Jumbo frames support (chained buffers)
• Take security seriously
• Multiple operation mode: ethernet, ip, punt/inject
• Lightweight library for apps - allows easy creation of applications which communicate over memif

It needs to be fast, but performance is not a number 1 priority.
memif – Security

• Point-to-point Master/Slave concept:
  • **Master** - Never exposes memory to slave
  • **Slave** - Responsible for allocation and sharing memory region(s) to Master
    • Slave can decide if it will expose internal buffers to master or copy data into separate shared memory buffer

• Shared memory data structures (rings, descriptors) are pointer-free
• Interfaces are always point-to-point, between master-slave pair
• Shared memory is initialized on connect and freed on disconnect
• Interface is uniquely identified by unix socket filename and interface id pair
• There is optional shared secret support per interface
• Optionally master can get PID, UID, GID for each connection to socket listener

Memory copy is a MUST for security.
memif – Control Channel

- Implemented as Unix Socket connection (AF_UNIX)
- Master is socket listener (allows multiple connections on single listener)
- Slave connects to socket
- Communication is done with fixed size messages (128 bytes):
  - **HELLO** (m2s): announce info about Master
  - **INIT** (s2m): starts interface initialization
  - **ADD_REGION** (s2m): shares memory region with master (FD passed in ancillary data)
  - **ADD_RING** (s2m): shares ring information with master (size, offset in mem region, interrupt eventfd)
  - **CONNECT** (s2m): request interface state to be changed to connected
  - **CONNECTED** (m2s): notify slave that interface is connected
  - **DISCONNECT** (m2s, s2m): disconnect interface
  - **ACK** (m2s, s2m): Acknowledge
memif – Shared Memory layout
memif – Shared Memory layout

• Rings and buffers in shared memory are referenced with (region_index, offset) pair
  • Much easier to deal with SEGFAULTS caused by eventual memory corruption
• Slave shares one or more memory regions with master by passing mmap() file descriptor and region size information (ADD_REGION message)
• Slave initializes rings and descriptors and shares their location (region_index, offset), size, direction and efd with master (ADD_RING) message
• Each ring contains header and array of buffer descriptors
  • number of descriptors is always power-of-2 for performance reasons (1024 as default)
• Buffer descriptor is 16 byte data structure which contains:
  • flags (2byte) – space for various flags, currently only used for buffer chaining
  • region_index (2 byte) – memory region where buffer is located
  • offset (4 bytes) – buffer start offset in particular memory region
  • length (4 byte) – length of actual data in the buffer
  • metadata (4 byte) – custom use space
Memif Performance – L2

**Packet Operations**
- Network Functions
- Virtual I/O Memory Copy

**Network Functions**
- I/O Device Memory Copy
- Link Rx/Tx
- Test Packet Tx/Rx

**Note:** packets are passing “vswitch, vrouter” DUT twice per direction, so the external throughput numbers reported in the table should be doubled to get per CPU core throughput.

**Packet Throughput**
- **Packet Throughput** [Mpps]
  - **64B**
    - Hsw_noTB: 7.0
    - Skx_noTB: 8.5
    - Skx_TB: 10.9
  - **IMIX**
    - Hsw_noTB: 5.2
    - Skx_noTB: 6.1
    - Skx_TB: 7.5
  - **1518B**
    - Hsw_noTB: 2.0
    - Skx_noTB: 2.6
    - Skx_TB: 3.0

**Bandwidth Throughput**
- **Bandwidth Throughput** [Gbps]
  - **64B**
    - Hsw_noTB: 4.7
    - Skx_noTB: 5.7
    - Skx_TB: 7.3
  - **IMIX**
    - Hsw_noTB: 15.5
    - Skx_noTB: 18.2
    - Skx_TB: 22.2
  - **1518B**
    - Hsw_noTB: 25.2
    - Skx_noTB: 32.1
    - Skx_TB: 36.5

* Maximum Receive Rate (MRR) Throughput - measured packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss.

**Hsw** – Intel Xeon® Haswell, E5-2699v3, 2.3GHz, noHT. Results scaled up to 2.5GHz and HT enabled.
**Skx** – Intel Xeon® Skylake, Platinum 8180, 2.5GHz, HT enabled.
**TB** – TurboBoost enabled.
**noTB** – TurboBoost disabled.
Memif Performance – IPv4

**Note:** packets are passing “vswitch, vrouter” DUT twice per direction, so the external throughput numbers reported in the table should be doubled to get per CPU core throughput.

<table>
<thead>
<tr>
<th>Packet Size</th>
<th>Packet Throughput*</th>
<th>Bandwidth Throughput*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Skx_noTB</td>
<td>Skx_TB</td>
</tr>
<tr>
<td>64B</td>
<td>6.15</td>
<td>7.32</td>
</tr>
<tr>
<td>IMIX</td>
<td>4.49</td>
<td>5.5</td>
</tr>
<tr>
<td>1518B</td>
<td>2.44</td>
<td>2.62</td>
</tr>
</tbody>
</table>

*Maximum Receive Rate (MRR) Throughput – measured packet forwarding rate under the maximum load offered by traffic generator over a set trial duration, regardless of packet loss.

Hsw – Intel Xeon® Haswell, E5-2699v3, 2.3GHz, noHT. Results scaled up to 2.5GHz and HT enabled.
Skx – Intel Xeon® Skylake, Platinum 8180, 2.5GHz, HT enabled.
TB – TurboBoost enabled.
noTB – TurboBoost disabled.
Summary

• **FD.io VPP** enables flexible software Network Functions
  On Bare-Metal, VMs and Containers
  High-performance

• **Ligato** manages lifecycle and topology of CNF services
  Enables network Service Function Chaining (SFC)
  Integrated with K8s

• **FD.io memif** is a virtual packet interface for Apps and Containers
  Optimised for performance (Mpps, Gbps, CPP* and IPC**)
  Safe and Secure, Zero memory copy on Slave side

• **Memif library for cloud-native Apps available**
  Allows easy integration for communicating over memif
  Potential to become a de facto standard.

* CPP, Cycles Per Packet
** IPC, Instructions per Cycle
THANK YOU!

Accelerating the Development of Cloud-native CNFs
Opportunities to Contribute

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!