Building high performance network functions in VPP

Ole Trøan, ot@cisco.com, VPP contributor
FOSDEM 2018
This talk?

• Goal: Make you into VPP developers

• Agenda:
  VPP architecture
  An example decomposed VPP VNF implementation
What’s this NFV malarkey anyway?
Approach #1

• Take your hardware appliance, port it to Linux, stuff it into a VM. Done.

• Disaggregation: Take each one of all the features on the router’s datapath, stuff it into a separate VM and invent some marvellous way of chaining them all back together.
Approach #2

• Decomposition / Disaggregation

• Deployment model:
  Bare-metal, VM, container, (unikernel)

• Control plane?
What is VPP?

• A framework for building forwarding functions.

• Written in C

• Multi-core, Portable. x86, ARM, PowerPC

• Data-plane / Control-plane separation
  High performance shared memory API
  Debug CLI

• Punt path to Linux kernel / Control plane apps.

• Tracing, logging, counters

• Scheduler, lightweight processes

• Host stack with own UDP/TCP implementation

• Drivers for: AF_PACKET, TAP, MEMIF, DPDK...
VPP highlights

- Vector packet processing. As opposed to scalar packet processing. A vector of packets are processed through a graph of forwarding functions. This ensures optimal usage of the instruction cache.

- Modern, lock-less scalable data structures. A use of indices instead of pointers for direct lookup and robustness (as opposed to data structures chasing pointers). Effective use of a modern CPUs memory hierarchy and modern CPU instructions (e.g. AVX2).

- Forwarding as a graph. Where each graph node is independent and does a limited amount of work on a vector of packets. Which ensures a very high hit rate on the instructions cache.

- Plugins as first-class citizens. Allows for a completely bespoke forwarding plane, by dynamically adjusting the forwarding graph. The graph can be custom built for the application, the software image and running application need only contain the set of graph nodes used. And the graph node can be developed using VPP libraries outside or inside of the main VPP software repository.

- Performance. Linear scale by number of cores. 2.8 instructions per cycle (on Intel Xeon Broadwell), very efficient use of a CPUs instruction cache and memory hierarchy.

- Robust and secure. VPP processes packets with a consistent latency and does not have a hierarchy of gradually slower and more feature complete forwarding paths (slow-path, fast-path as in IOS).

- Super programmable data plane. VPP is already integrated with ML2, ODL, ACI. It offers through its middleware agent interfaces over NETCONF/YANG, REST. It has direct language bindings in Python, Lua, Java and C. It is programmable over GRPC. Everything is done over APIs. The APIs are the first class citizens.

- New shared memory interfaces for container to container communication and from container to VPP
Universal Dataplane: Features

**Hardware Platforms**
- Pure Userspace: X86, ARM 32/64, Power, Raspberry Pi

**Interfaces**
- DPDK/Netmap/AE_Packet/TunTap
- Vhost-user: multi-queue, reconnect, Jumbo Frame Support

**Language Bindings**
- C/Java/Python/Lua

**Tunnels/Encaps**
- GRE/VXLAN/VXLANT-GE/LISP-GPE/NH IPSec
- Including HW offload when available

**MPLS**
- MPLS over Ethernet/GRE
- Deep label stacks supported

**Routing**
- IPv4/IPv6
- 14+ MPPS, single core
- Hierarchical FIBs
- Multimillion FIB entries
- Source RPF
- Thousands of VRFs
- Controlled cross-VRF lookups
- Multipath - ECMP and Unequal Cost

**Segment Routing**
- SR MPLS/IPv6
- Including Multicast

**Switching**
- VLAN Support
- Single/DL Double tag
- L2 forwarding w/ EFP
- Bridge Domain concept
- VTR - push/pop/Translate (1:1, 1:2, 1:2)
- L2 Learning - default limit of 50k table
- Bridging
- Split horizon group support/EFP Filtering
- Proxy ARP
- ARP termination
- IRB - BVI Support with RouterMac assignment
- Flooding
- Input ACLs
- Interface cross-connect
- L2 GRE over IPSec tunnels

**Network Services**
- DHCPv4 client/proxy
- DHCPv6 Proxy
- MAP/LW46 - IPv6aaS
- MagLev-like Load
- Identifier Locator Addressing
- NSH SFC SFF’s & NSH Proxy
- LDP
- BFD
- Policer
- Multiple million Classifiers - Arbitrary N-tuple

**Inband iOAM**
- Telemetry export infra (raw IPFIX)
- iOAM for VXLAN-GPE (NGENA)
- SRv6 and iOAM co-existence
- iOAM proxy mode / caching
- iOAM probe and responder

**LISP**
- LISP xTR/RTR
- L2 Overlays over LISP and GRE encaps
- Multitenancy
- Multihome
- Map/Resolver Failover
- Source/Dest control plane support
- Map-Register/Map-Notify/RLOC-probing

**Security**
- Mandatory Input Checks:
  - TTL expiration header checksum
  - L2 length < IP length
  - ARP resolution/snooping
  - ARP proxy
- SNAT
- Ingress Port Range Filtering
- Per interface whitelists
- Policy/Security Groups/GBP (Classifier)

**Monitoring**
- Simple Port Analyzer (SPAN)
- IP Flow Export (IPFIX)
- Counters for everything
- Lawful Intercept
Packet processing is decomposed into a directed graph of nodes. ... packets move through graph nodes in vector. ... graph nodes are optimized to fit inside the instruction cache. ... packets are pre-fetched into the data cache.

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

* Each graph node implements a “micro-NF”, a “micro-NetworkFunction” processing packets.
VNF: MAP/LW46
MAP VPP VNF
Done. Questions?

- Code: https://github.com/FDio/vpp

- MAP VNF: 100s of line of forwarding code. 1000000s line of control plane.

- Does it need to be so darn complicated?