How to Push Extreme Limits of Performance and Scale with Vector Packet Processing Technology

FD.io VPP Overview
Maciek Konstantynowicz, an Engineer
mkonstan@cisco.com
maciek@cisco.com
Evolution of Programmable Networking

• Many industries are transitioning to a more dynamic model to deliver network services

• The great unsolved problem is how to deliver network services in this more dynamic environment

• Inordinate attention has been focused on the non-local network control plane (controllers)
  • Necessary, but insufficient

• There is a giant gap in the capabilities that foster delivery of dynamic Data Plane Services
FD.io - Fast Data input/output – for Internet Packets and Services

What is it about – continuing the evolution of Computers and Networks:

- **Computers** => **Networks** => **Networks of Computers** => **Internet of Computers**
- **Networks** in Computers – Requires efficient packet processing in Computers
- Enabling modular and scalable Internet packet services in the Cloud of Computers – routing, bridging, tunneling and servicing packets in CPUs
- Making Computers be part of the Network, making Computers become a-bigger-helping-of-Internet

FD.io: [www.fd.io](http://www.fd.io)
Aside: How does it fit into building the new Internet infrastructure.. An example - CleanSlate build “TeraStream”

RIPE67 – talk by Peter Lothberg “TeraStream – A Simplified IP Network Service Delivery Model”

https://ripe67.ripe.net/presentations/131-ripe2-2.pdf


“Software Defined X” movement

Software Defined Operators

Software Defined Networking

...
Introducing Fast Data: fd.io

• New project in Linux Foundation
  • Multi-party
  • Multi-project

• What does multi-party mean?
  • Multiple members - Open to all

• What does multi-project mean?
  • Multiple subprojects
  • Subproject autonomy
  • Cross project synergy
  • Open to new subprojects
  • Anyone can propose a subproject
  • Allows for innovation

Create a Platform that enables Data Plane Services that are:
  • Highly performant
  • Modular and extensible
  • Open source
  • Interoperable
  • Multi-Vendor

Platform fosters innovation and synergistic interoperability between Data Plane Services

Source of Continuous Integration resources for Data Plane services based on the Consortium’s project/subprojects

Meet the functionality needs of developers, deployers, datacenter operators

www.fd.io  git.fd.io  jira.fd.io
wiki.fd.io  gerrit.fd.io  jenkins.fd.io
Introducing Vector Packet Processor - VPP

- VPP is a rapid packet processing development platform for highly performing network applications.
- It runs on commodity CPUs and leverages DPDK.
- It creates a vector of packet indices and processes them using a directed graph of nodes – resulting in a highly performant solution.
- Runs as a Linux user-space application.
- Ships as part of both embedded & server products, in volume.
- Active development since 2002.
VPP Architecture - Modularity Enabling Flexible Plugins

Plugins == Subprojects

Plugins can:
- Introduce new graph nodes
- Rearrange packet processing graph
- Can be built independently of VPP source tree
- Can be added at runtime (drop into plugin directory)
- All in user space

Enabling:
- Ability to take advantage of diverse hardware when present
- Support for multiple processor architectures (x86, ARM, PPC)
- Few dependencies on the OS (clib) allowing easier ports to other Oses/Env
## VPP Feature Summary

**IPv4/IPv6**
- 14+ MPPS, single core
- Multimillion entry FIBs
- Source RPF
- Thousands of VRFs
  - Controlled cross-VRF lookups
- Multipath – ECMP and Unequal Cost
- Multiple million Classifiers – Arbitrary N-tuple
- VLAN Support – Single/Double tag

**Counters for everything**

Mandatory Input Checks:
- TTL expiration
- header checksum
- L2 length < IP length
- ARP resolution/snooping
- ARP proxy

**IPv4**
- GRE, MPLS-GRE, NSH-GRE, VXLAN
- IPSEC
- DHCP client/proxy
- CG NAT

**IPv6**
- Neighbor discovery
- Router Advertisement
- DHCPv6 Proxy
- L2TPv3
- Segment Routing
- MAP/LW46 – IPv4aas
- iOAM

**MPLS**
- MPLS-o-Ethernet – Deep label stacks
  - supported

**L2**
- VLAN Support
  - Single/Double tag
- L2 forwarding with
  - EFP/BridgeDomain concepts
- VTR – push/pop/Translate
  - (1:1, 1:2, 2:1, 2:2)
- Mac Learning – default limit of 50k addresses
- Bridging – Split-horizon group support/EFP Filtering
- Proxy Arp
- Arp termination
- IRB – BVI Support with
  - RouterMac assignment
- Flooding
- Input ACLs
- Interface cross-connect
Network workloads vs. compute workloads

- **They are just a little BIT different**
  - They are all about processing packets
  - At 10GE, 64B frames can arrive at 14.88Mfps – that’s 67 nsec per frame.
  - With 2GHz CPU core each clock cycle is 0.5 nsec – that’s 134 clock cycles per frame.
  - BUT it takes ~70 nsec to access memory – not much time to do work if waiting for memory access.

- **Efficiency of dealing with packets within the computer is essential**
  - Moving packets:
    - Packets arrive on physical interfaces (NICs) and virtual interfaces (VNFs) - need CPU optimized drivers for both.
    - Drivers and buffer management software must not rely on memory access – see time budget above.
  - Processing packets:
    - Header manipulation, encaps/decaps, lookups, classifiers, counters.
    - Need packet processing optimized for CPU platforms

- **CONCLUSION - Need to pay attention to Computer efficiency for Network workloads**
  - Computer efficiency for x86-64 = close to optimal use of x86-64 uarchitectures: core and uncore resources
What is Vector Packet Processing?

- High performance packet-processing stack for commodity CPUs
  - x86_64, ppc-64-BE, aarch64-LE
  - Endian clean, 32 / 64-bit clean
  - Linux user-mode process
  - Leverage DPDK, widely-available kernel modules
    - (uio, igb_uio, uio_pci_generic)

- Linux user-space
  - Same image works in a VM, over a host kernel, in an LXC
  - Physical NICs via PCI direct-map

- Active development since 2002

- Ships as part of Cisco embedded and server products, in volume
Packet Processing on Commodity Hardware

- Packet-processing: load/store-intensive, big tables, N-tuple problems
- PP on CH: significantly different than PP on NPUs
  - NPU: e.g. 2048 outstanding prefaches, SRAM
  - Commodity HW: 8 → 16 outstanding prefaches, DDRn
  - NPU: thousands of PPEs processing single packets
  - Commodity HW: tens of general-purpose cores
  - NPU: work distributor, TCAM, specialized counter support, QoS / queueing support
- VPP solves these problems—or a useful subset—on commodity hardware
  - Structure the computation for CH’s convenience
Scalar Packet Processing

- A fancy name for processing one packet at a time
  - Traditional, straightforward implementation scheme
  - Interrupt, a calls b calls c … return return return RFI
  - Considerable stack depth

- Issue #1: thrashing the I-cache
  - When code path length exceeds the primary I-cache size, each packet incurs an identical set of I-cache misses
  - Only workaround: bigger caches
Scalar Packet Processing, cont’d

- Dependent read latency on big forwarding tables
  - Example: 4 x 8 mtrie walk. Assume tables do not fit in cache.
  - Lookup 5.6.7.8: read root_ply[5], then ply_2[6], the ply_3[7], the ply_4[8]
  - Big tables: reads stall for ~170 clocks
- Few opportunities to mitigate (“prefetch around”) read latency stalls
Vector Packet Processing

- Process more than one packet at a time
  - Grab all available packets from device RX ring
  - Form a “frame” (vector) comprising packet indices in RX order
  - Process frames using a directed graph of nodes
- What does this reorganization accomplish?
  - Fixes the I-cache thrashing problem
  - Allows us to mitigate the dependent read latency problem
  - “Circuit time” reaches a (stable) equilibrium value based on offered load
Vector Packet Processing I-Cache Effect

- Since each graph node processes more than one packet
  - First packet warms up the I-cache
  - All other packets hit in the I-cache
  - Warm up the CPU branch predictor
- Node dispatch functions typically comprise ~150 lines of code
  - First choice: f(x) fits in the L0 I-cache
  - Second choice: multiple foreach_vector_element passes
  - Third choice: use multiple nodes
Graph Scheduler

- Always process vector elements in order
- Run-to-completion, until all vector elements have been disposed of
- As vector size increases, processing cost per pkt decreases
  - Amortize I-cache misses over a larger N
- Stable vector size equilibrium
  - Start with vector size in equilibrium
  - Add a delay: clock-tick interrupt or similar
  - Vector size increases, but cost/pkt decreases
  - Computation returns to the previous equilibrium vector size
  - Consistent per-packet latency
Exploiting Multiple Cores

• Embarrassing parallelism
  • Ingress flow hashing (e.g. h/w RSS hash onto multiple RX queues)
  • N x worker threads: node graph replicas
  • H/W-permitting: N x TX queues
  • Easy to spin up, configure problem-specific threads

• Counters
  • Unsigned short per-worker-thread non-atomic counters
  • Overflow: 64-bit atomic add
Control-Plane Binary APIs

- API generator—yacc + hand-rolled lexical analyzer
  - Header file: API version signature, message enumeration, C typedefs, etc.
- Two transport types
  - Unidirectional queues in shared memory, stream sockets
- All message data in network byte order
- Asynchronous messaging, per-message “context” opaque
- Program ~700k routes/second into the ip4 FIB
- Client library: connect / disconnect from the data plane
- Plugins can / ordinarily do define their own control-plane APIs
- Trace, pretty-print, scriptify, replay long sequences. Post-mortem capture.
So what does this mean?
FD.io Design Engineering by Benchmarking
Continuous Performance Lab (CPL)

Fully automated testing infrastructure
- Covers both programmability and data planes
- Code breakage and performance degradations identified before patch review
- Review, commit and release resource protected

Continuous Functional Testing
- Virtual testbeds with network topologies
- Continuous verification of functional conformance
- Highly parallel test execution

Continuous Software and Hardware Benchmarking
- Server based hardware testbeds
- Continuous integration process with real hardware verification
  - Server models, CPU models, NIC models

Implemented in FD.io CSIT project
FD.io Continuous Performance Lab
a.k.a. The CSIT Project (Continuous System Integration and Testing)

- **What it is all about – CSIT aspirations**
  - **FD.io VPP benchmarking**
    - VPP functionality per specifications ([RFCs](#))
    - VPP performance and efficiency ([PPS](#), [CPP](#))
      - Network data plane - throughput Non-Drop Rate, bandwidth, PPS, packet delay
      - Network Control Plane, Management Plane Interactions (memory leaks!)
    - Performance baseline references for HW + SW stack ([PPS](#), [CPP](#))
    - Range of deterministic operation for HW + SW stack ([SLA](#))
  - **Provide testing platform and tools to FD.io VPP dev and usr community**
    - Automated functional and performance tests
    - Automated telemetry feedback with **conformance**, **performance** and **efficiency** metrics
  - **Help to drive good practice and engineering discipline into FD.io VPP dev community**
    - Drive innovative optimizations into the source code – verify they work
    - Enable innovative functional, performance and efficiency
    - Make progress faster
    - Prevent unnecessary code “harm”

Legend:
1 RFC – Request For Comments – IETF Specs basically
2 PPS – Packets Per Second
3 CPP – Cycles Per Packet (metric of packet processing efficiency)
4 SLA – Service Level Agreement
CSIT/VPP-v16.06 Report

1 Introduction
2 Functional tests description
3 Performance tests description
4 Functional tests environment
5 Performance tests environment
6 Functional tests results
6.1 L2 Bridge-Domain
6.2 L2 Cross-Connect
6.3 Tagging
6.4 VXLAN
6.5 IPv4 Routing
6.6 DHCPv4
6.7 IPv6 Routing
6.8 COP Address Security
6.9 GRE Tunnel
6.10 LISP
7 Performance tests results
7.1 VPP Trend Graphs RFC2544:NDR
7.2 VPP Trend Graphs RFC2544:PDR
7.3 Long Performance Tests - NDR and PDR Search
7.4 Short Performance Tests - ref-NDR Verification

- Testing coverage summary
  - L2, IPv4, IPv6
  - Tunneling
  - Stateless security
- Non Drop Rate Throughput
  - 8Mpps to 10Mpps per CPU core at 2.3GHz*
  - No HyperThreading
- Improvements since v16.06
  - 10Mpps to 12Mpps per CPU core at 2.3GHz*
  - With HyperThreading gain ~10%
  - See next slides


### Compute Node Hardware

<table>
<thead>
<tr>
<th>Compute Node Hardware</th>
<th>UCS C240 M4SX – 2-CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>Intel® C610 series chipset</td>
</tr>
<tr>
<td>CPU</td>
<td>2x Intel® Xeon® Processor E5-2698 v3 (each with 16 CPU cores, clocked at 2.6GHz, 40MB Last Level Cache)</td>
</tr>
<tr>
<td>Memory</td>
<td>2133 MHz, 128 GB Total</td>
</tr>
<tr>
<td>NICs</td>
<td>6x 2p40GE Intel® XL710 Network Interface Cards 12x 40GE = 480GE of external network I/O</td>
</tr>
</tbody>
</table>

### Compute Node Software

<table>
<thead>
<tr>
<th>Compute Node Software</th>
<th>Version</th>
</tr>
</thead>
</table>
| Host Operating System | Ubuntu 14.04.3 LTS  
Kernel version: 3.13.0-63-generic |
| DPDK                  | DPDK 16.4 |
| FD.io VPP             | VPP v16.06 |

*The Fast Data Project (FD.io)*

A Computer...
Running a Network Data Plane

A Computer...
Being a Router

A Computer...
Being part of the Internet...
... An integral part thereof!
VPP-based vSwitch in Phy-VS-Phy Topology
Scenario: Comput Host with 12x 40GE interfaces, on 6 NICs
VPP Scaling Up The Packet Throughput across 40GE interfaces, PCIe Gen3 slots and CPU cores

Non Drop Rate Throughput* - Zero Packet Loss - Gigabits per second [Gbps]

- FD.io VPP data plane IPv4 routing on UCS C240-M4 – 2x CPU 2.3GHz, 6x 2p40GE NIC
- FD.io VPP supports multi-threading with linear scaling of Mpps and Gbps with adding cores and VPP threads

- With one CPU core per 40GE interface – NIC Gbps max-rate for IMIX and above frame sizes
- With two CPU cores per 40GE interface – NIC Gbps max-rate for 128B and above frame sizes

*Throughput benchmarks with VPP running IPv4 routed-forwarding with 1M IPv4/32 routes

**IMIX frame size = (7x 64B, 4x 570B, 1x 1518B), avg. 353.8Bytes
VPP Scaling Up The Packet Throughput across 40GE interfaces, PCIe Gen3 slots and CPU cores

Non Drop Rate Throughput* - Zero Packet Loss – Mega packets per second [Mpps]

- FD.io VPP data plane IPv4 routing on UCS C240-M4 – 2x CPU 2.3GHz, 6x 2p40GE NIC
- FD.io VPP supports multi-threading with linear scaling of Mpps and Gbps with adding cores and VPP threads

With one CPU core per 40GE interface – NIC Mpps max-rate for IMIX and above frame sizes

With two CPU cores per 40GE interface – NIC Mpps max-rate for 64B and above frame sizes

*Throughput benchmarks with VPP running IPv4 routed-forwarding with 1M IPv4 /32 routes

** IMIX frame size = (7x 64B, 4x 570B, 1x 1518B), avg. 353.8Bytes
FD.io VPP – A Platform for Interesting Work ...

• Modern IP router data plane out-of-the-box
  • Advanced, modular, scales, optimized SW-HW interface
  • A platform to build on
• All sorts of crypto and tunneling things
• ILA at IETF96 Hackathon in Berlin
• Telemetry – apps, users, flows, ...
• Modern TCP stack anyone?
  • E.g. TCP ex-machina, Keith Winstein https://github.com/tcpexmachina/remy
• ...

Next Steps – Get Involved

We invite you to Participate in fd.io

- [Get the Code, Build the Code, Run the Code](#)
- [Read/Watch the Tutorials](#)
- [Join the Mailing Lists](#)
- [Join the IRC Channels](#)
- [Explore the wiki](#)
- [Join fd.io as a member](#)
Q&A
A Parable to Read ...

• A short history of a system design use case
  • A railway company designing a train system
  • A manuscript written in the middle of year 1973
  • Published in year 1976 – was still applicable
  • Cited later today (year 2012, 2016) – and still applies

• URL link
  • http://www.cs.utexas.edu/users/EWD/transcriptions/EWD05xx/EWD594.html

• Authored by
  • http://translate.google.com/#nl|en|Edsger%20Wybe%20Dijkstra
  • http://en.wikipedia.org/wiki/Edsger_W._Dijkstra